Laboratory ~~8~~ 7

Confusion matrices

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

FINO AD ORA ABBIAMO PRESO SEMPRE QUELLI CON LA MASSIMA PROBABILITÀ A POSTERIORI

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

NEL DATASET IRIS CI SONO POCHI ERRORI QUINDI NON È INTERESSANTE DA VEDERE

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 log-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

PER CALCOLARE LA MATRICE DI CONFUSIONE DEVI CALCOLARE LA PROB A POSTERIORI

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

UNIFORM COST MATRIX

MATRICE NCLASS X N CLASS NUMERO DELLE CLASSI OGNI ELEMENTO x_{ij} RAPPRESENTA IL COSTO DI CLASSIFICARE UN ELEMENTO DI CLASSE i COME j. IN QUESTA MATRICE IL COSTO DI CLASSIFICARE CON LA GIUSTA CLASSE VALE 0. MENTRE È 1 PER GLI ALTRI

	0	1	2
0	0	1	1
1	1	0	1
2	1	1	0

COSTO GIUSTA CLASSIFICAZIONE

Optimal Bayes decision

DICE SOLO COME FUNZIONA

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 $k = 1 \ldots K$, in the following we use indices $k = 0 \ldots K-1$. 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

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

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. CIOÈ QUANDO STAI SBAGLIANDO LA CLASSIFICAZIONE

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:

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

LA CLASSE È QUELLA CHE HA IL MINIMO COSTO DI BAYES

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

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

LLR È IL RAPPORTO DI VEROSOMIGLIANZA LOGARITMICO È UTILE NEI TEST DI IPOTESI E IN APPLICAZIONI DI RICONOSCIMENTO DI MODELLO E CLASSIFICAZIONE

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

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

$$LLR = \log\left(\frac{P(D \mid \theta_1)}{P(D \mid \theta_0)}\right)$$

DATI D

MODELLO ALTERNATIVO

MODELLO NULLO

- **VALORE POSITIVO**: UN VALORE POSITIVO I DATI SONO PIÚ PROBABILI SOTTO IL MODELLO ALTERNATIVO
- VALORE NEGATIVO: PIÚ PROB SOTTO MODELLO NULLO
- VALORE ZERO: VALORI UGUALMENTE PROB SOTTO ENTRAMBI I MODELLI

TEST DI IPOTESI: SI USA PER DECIDERE TRA H_0 O H_1. SI CONFRONTA LLR CON UNA SOGLIA PER DIRE SE È H_0 O H_1

↳ OGNI ELEMENTO CONTIENE LA PROB CHE APPARTENGA A QUELLA CLASSE L'ELEMENTO

H_T → INFERNO

H_F → PARADISO

(π_1, C_{fn}, C_{fp})

PRIORITÀ

COSTO FALSO NEGATIVO

COSTO FALSO POSITIVO

COMPUTE_OPTIMAL_BAYES_BINARY_LLR

- CALCOLA UNA SOGLIA th COME IL LOGARITMO TRA PRIOR · C_{fn} · C_{fp} SERVE PER DECIDERE TRA DUE CLASSI. POI USI LA th PER VEDERE A CUI APPARTIENE QUEL VALORE IN BASE ALLA SOGLIA and to prior class probabilities $P(C = \mathcal{H}_T) = \pi_1$, $P(C = \mathcal{H}_F) = 1 - \pi_1$.

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.

PRIOR (π)
È LA PROBABILITÀ A PRIORI DI UNA CLASSE IN UN COMPITO DI CLASSIFICAZIONE BINARIA.
È UTILIZZATA PER CALCOLARE LA SOGLIA.

Prediction \ Class	0	1
0	293 ($M_{0,0}$)	96 ($M_{1,0}$)
1	109 ($M_{0,1}$)	304 ($M_{1,1}$)

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

Prediction \ Class	0	1
0	271	80
1	131	320

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

Prediction \ Class	0	1
0	257	75 fn
1	145 fp	325

DIMINUISCE
AUMENTA

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

Prediction \ Class	0	1
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 opposite is true when $C_{1,0}$ is higher.

• SE $C_{0,1}$ CRESCE CI SONO PIÙ C_{fp} E MENO C_{fn}
• L'OPPOSTO QUANDO $C_{1,0}$ È ALTO

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

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

COME ESPRIMI IL RISCHIO

where c_i is the actual class for sample i and c_i^* is the predicted class for the same sample. P_{fn} and P_{fp} are the false negative and false positive 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

NUMERO DEI SAMPLE PREDETTI COME j MA CHE APPARTENGONO AD i

$$
P_{fn} = \frac{FN}{FN + TP} = \frac{M_{0,1}}{M_{0,1} + M_{1,1}} \,, \quad P_{fp} = \frac{FP}{FP + TN} = \frac{M_{1,0}}{M_{0,0} + M_{1,0}}
$$

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

BINARY TASK : EVALUATION

$$DCF_u = B = \pi_1 C_{fn} P_{fn} + (1-\pi_1) C_{fp} P_{fp}$$

$$P_{fn} = \frac{M_{0,1}}{M_{0,1} + M_{1,1}} \qquad P_{fp} = \frac{M_{1,0}}{M_{0,0} + M_{1,0}}$$

CLASS

$$\text{PREDICT} \quad \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & M_{0,0} & fn\ M_{0,1} \\ 1 & fp\ M_{1,0} & M_{1,1} \end{array}$$

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

→ PER CALCOLARLO $\frac{DCF}{B_{dummy}}$

CALCOLA IL COSTO MINIMO POSSIBILE CHE SI VERIFICA QUANDO TUTTI GLI ESEMPI SONO CLASSIFICATI CORRETTAMENTE

OTTIENI UNA MISURA DI ERRORE TRA 0 E 1

QUESTA NORMALIZZAZIONE È UTILE PER CONFRONTARE LE PRESTAZIONI DI DIVERSI CLASSIFICATORI OPPURE PER CONFRONTARE LE PRESTAZIONI SULLO STESSO SET DI DATI MA CON DIVERSA PRIORITÀ E COSTI

(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

BAYES RISK CI PERMETTE DI CONFRONTARE DIVERSI SISTEMI POSSIAMO CALCOLARE UN COSTO DI RICONOSCIMENTO NORMALIZZATO DIVIDENDO IL RISCHIO DI BAYES PER IL RISCHIO DI UN SISTEMA OTTIMALE CHE NON UTILIZZA DATI DI TEST

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

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

(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

NON VANNO BENE

SE IL VALORE È VICINO A ZERO INDICA CHE IL CLASSIFICATORE HA COMMESSO POCHI ERRORI RISPETTO AL COSTO MINIMO POSSIBILE. VICINO AD 1 INVECE VUOL DIRE CHE HA COMMESSO MOLTI ERRORI

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

QUESTO NON VIENE APPLICATO AL MULTICLASS

Minimum detection costs

PUOI CALCOLARE LO SCORE PER UNA CATTIVA SEPARAZIONE DELLE CLASSI E PER UNA CATTIVA CALIBRAZIONE DEI PUNTEGGI

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).

PER RICALIBRARLI PUOI USARE UN PICCOLO INSIEME DI DATI ETICHETTATI CHE SONO DIVERSI DA QUELLI CHE SONO STATI UTILIZZATI IN ADDESTRAMENTO

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). → IN ALTERNATIVA PUOI CALCOLARE LA SOGLIA SULLO STESSO SET DI VALIDAZIONE E UTILIZZARE LA SOGLIA PER LA POPOLAZIONE DI TEST

DEVI CALCOLARE IL DCF MINIMO QUESTO IMPLICA L'ITERAZIONE SU TUTTE LE POSSIBILI SOGLIE CALCOLANDO LA MATRICE DI CONFUSIONE E IL DCF PER CIASCUNA E INFINE SELEZIONA IL MINIMO DCF

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 (notice that the DCF can change only when we change a prediction, and that can happen only when the threshold moves "across" one of the evaluation scores). 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})	$\boxed{\text{min DCF}}$
(0.5, 1, 1)	0.506
(0.8, 1, 1)	0.752
(0.5, 10, 1)	0.842
(0.8, 1, 10)	0.709

PER OGNI SOGLIA t SI CALCOLA LA MATRICE DI CONFUSIONE CHE SI OTTERREBBE UTILIZZANDO t E NORMALIZZANDO

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 classifier 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.

PER CALCOLARE IL MINIMO DCF

- LLR → UN ARRAY DI PUNTEGGI LLR
- CLASS LABEL → ETICHETTA DEI DATI
- PRIOR, Cfn, Cfp

LA FUNZIONE ITERA SU TUTTE LE POSSIBILI SOGLIE CHE SONO I PUNTEGGI LLR PIÚ -INF E +INF.
CALCOLA LA PREVISIONE PER OGNI SOGLIA E CALCOLA IL DCF PER LE PREVISIONI TENENDO TRACCIA DEL MINIMO DCF E DELLA CORRISPONDENTE SOGLIA.
DEVI CALCOLARE TUTTI I DCF E CHE SOGLIA TI HA DATO IL MINIMO

Esatto, la soglia restituita dalla funzione compute_minDCF_binary_slow è il valore di Log-Likelihood Ratio (LLR) che minimizza il Detection Cost Function (DCF) per il tuo modello di classificazione binaria. Questa soglia è utilizzata per decidere se un esempio deve essere classificato come positivo o negativo. Nel contesto della classificazione binaria, se il punteggio LLR di un esempio è superiore alla soglia, l'esempio viene classificato come positivo; altrimenti, viene classificato come negativo. Quindi, questa soglia è il punto di decisione ottimale tra le due classi, in termini di minimizzazione del costo totale della classificazione errata.

- VERSIONE FAST

La funzione compute_Pfn_Pfp_allThresholds_fast calcola le probabilità di falso negativo (Pfn) e falso positivo (Pfp) per tutte le possibili soglie in un modo più efficiente rispetto alla versione "slow". Nella versione "slow", per ogni soglia possibile, le previsioni e le matrici di confusione vengono ricalcolate da zero. Questo può essere molto costoso in termini di tempo se ci sono molte soglie possibili. Nella versione "fast", invece, i punteggi LLR e le etichette di classe vengono ordinati all'inizio. Poi, mentre si scorrono i punteggi LLR ordinati, si tiene traccia del numero di falsi negativi e falsi positivi e si aggiornano questi numeri ogni volta che si sposta la soglia. Questo approccio è più efficiente perché non richiede il ricalcolo delle previsioni e delle matrici di confusione per ogni soglia. Inoltre, la versione "fast" implementa un'ottimizzazione aggiuntiva per gestire i punteggi LLR ripetuti: invece di calcolare Pfn e Pfp per ogni punteggio LLR ripetuto, mantiene solo il valore che corrisponde a un effettivo cambiamento della soglia. Questo può ridurre ulteriormente il tempo di calcolo se ci sono molti punteggi LLR ripetuti

VIENE TORNATA LA PROB DI FALSO POSITIVO, DI FALSO NEGATIVO E LA SOGLIA CORRISPONDENTE
POI CALCOLI DCF PER TUTTI E PRENDI SOLO QUELLO MINIMO E PRENDI LA SOGLIA.
INVECE NEL METODO SLOW DOVEVI RICALCOLARE TUTTO OGNI VOLTA APPESANTENDO MOLTO IN BASE AL NUMERO DI PUNTEGGI CHE HAI.

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.

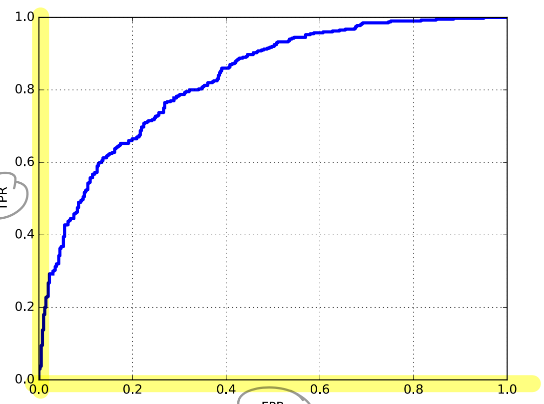
I GRAFICI NON TENGONO CONTO DEGLI ERRORI DOVUTI A UNA CATTIVA SELEZIONE DELLA SOGLIA POICHÉ SEMPLICEMENTE TRACCIANO COME CAMBIANO I TASSI DI ERRORE QUANDO MODIFICHI LA SOGLIA SUL SET DI VALUTAZ.

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 $P_{tp} = 1 - P_{fn}$. The ROC curve consists of points $(P_{tp}(t), P_{fp}(t))$ where t is the threshold. By sweeping all possible thresholds we can obtain the coordinates of the ROC curve.

COME VARIA IL NUMERO DI TRUE POSITIVE E FALSE POSITIVE AL VARIARE DELLA SOGLIA

Plot the ROC curve for the inferno-vs-paradiso scores. For each threshold, compute the confusion matrix and extract the P_{fn} and P_{fp} as you did in the previous section. Compute P_{tp} as $1 - P_{fn}$. Plot the curve that contains, on x-axis, all the false positive rates, and on the y-axis all corresponding true positive rates. You should obtain:

P_{tp}



P_{fp}

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])
```

DEVI TRACCIARE UN GRAFICO DELL'ERRORE DI BAYES IN FUNZIONE DI UN PRIORE $\tilde{\pi}$. SI PERMETTE DI GIUDICARE IL CLASSIFICATORE IN BASE AI LOG-ODDS PRECEDENTI $\tilde{p}$

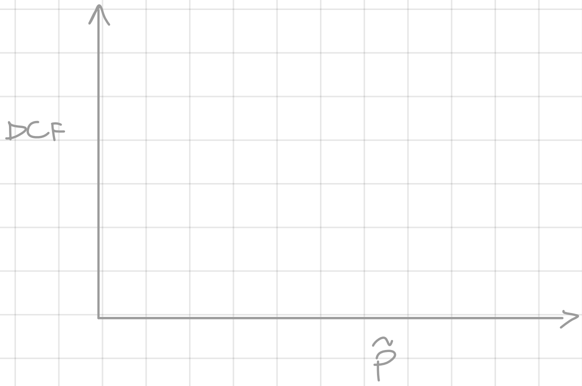
1) GENERA VALORI DI LOG-ODDS PRECEDENTI $\tilde{p}$. numpy.linspace(-3, 3, 21)

21 VALORI DISTRIBUITI DI $\tilde{p}$

2) $\forall\ \tilde{p}$ CALCOLI LA PRIOR $\tilde{\pi} = \frac{1}{1+e^{-p}}$

3) $\forall\ \tilde{\pi}$ SI DEVE CALCOLARE IL <u>DCF NORMALIZZATO</u> E IL <u>DCF MINIMO NORMALIZZATO</u> PER $(\tilde{\pi}, 1, 1)$

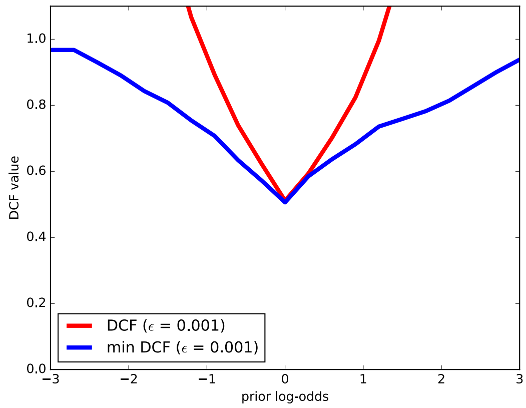
4) TRACCIA I VALORI DI DCF OTTENUTI IN FUNZIONE DEL LOG-ODDS $\tilde{p}$



QUESTO PERMETTE DI VALUTARE COME LE PRESTAZIONI DEL CLASSIFICATORE CAMBIA, CIOÈ AL VARIARE DI $\tilde{p}$

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

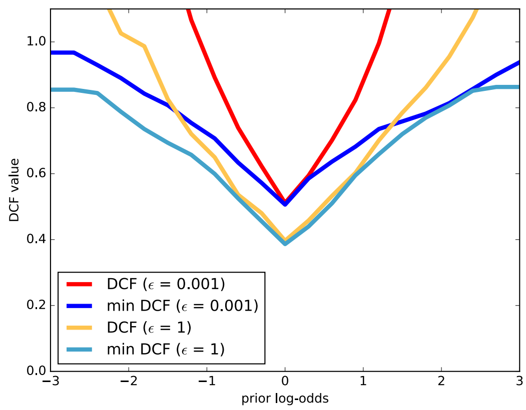
We now employ the tools that we have seen to compare the recognizer obtained with pseudocounts $\varepsilon = 0.001$ and $\varepsilon = 1$ (see Laboratory 6).

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:

IL VALORE EPSILON STA PER LIVELLO DI RUMORE O PERTURBAZIONE. SE È PIÙ ALTO QUESTO PUÒ AIUTARE A PREVENIRE L'OVERFITTING MENTRE SE È PIÙ PICCOLO C'È UN LIVELLO DI RUMORE O PERTURBAZIONE PIÙ BASSO.

	$\varepsilon = 0.001$		$\varepsilon = 1.0$	
(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

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).

SI PUÒ CONCLUDERE CHE IL MODELLO CON $\varepsilon = 1,0$ È SUPERIORE IN AMPIA GAMMA E PRODUCE PUNTEGGI CALIBRATI MEGLIO

Multiclass evaluation

QUELLO CHE SERVE PER IL PROGETTO

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 log-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 $\boxed{\pi_1 + \pi_2 + \pi_3 = 1\ .}$

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

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

$\boldsymbol{C}$ 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

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

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

Project

RILEVAMENTO DELLO SPOOFING DELLE IMPRONTE DIGITALI

Analyze the performance of the MVG classifier and its variants for different applications.

Start considering five applications, given by ($\overset{\text{PRIOR}}{\pi_1}, C_{fn}, C_{fp}$):

π PRIOR È LA PROBABILITÀ CHE UN CAMPIONE SIA CLASSIFICATO COSÌ A PRIORI. NEL CASO PER ESEMPIO DI π = 0.9 È PIÙ PROBABILE CHE L'IMPRONTA SIA LEGITTIMA.

- $(0.5, 1.0, 1.0)$, i.e., uniform prior and costs
  ↳ PROB SIA TRUE O FALSE È UGUALE
- $(0.9, 1.0, 1.0)$, i.e., the prior probability of a genuine sample is higher (in our application, most users are legit)
  → BASSA LA PROBABILITÀ CHE SIA AUTENTICA
- $(0.1, 1.0, 1.0)$, i.e., the prior probability of a fake sample is higher (in our application, most users are impostors)
- $(0.5, 1.0, 9.0)$, i.e., the prior is uniform (same probability of a legit and fake sample), but the cost of accepting a fake image is larger (granting access to an impostor has a higher cost than labeling as impostor a legit user - we aim for strong security)
- $(0.5, 9.0, 1.0)$, i.e., the prior is uniform (same probability of a legit and fake sample), but the cost of rejecting a legit image is larger (granting access to an impostor has a lower cost than labeling a legit user as impostor - we aim for ease of use for legit users)

Represent the applications in terms of effective prior. What do you obtain? Observe how the costs of mis-classifications are reflected in the prior: stronger security (higher false positive cost) corresponds to an equivalent lower prior probability of a legit user.

PER AVERE BUONI RISULTATI DEVONO ESSERE PIÚ VICINI POSSIBILE A ZERO

PRIOR 0.50 , COST FALSE NEGATIVE 1.00 , COST FALSE POSITIVE 1.00

MVG
Confusion Matrix:
[[927 75]
[65 933]]
DCF: 0.069964
DCF normalized: 0.139929
minDFC 0.13016833077316947

TIED
Confusion Matrix:
[[898 92]
[94 916]]
DCF: 0.093014
DCF normalized: 0.186028
minDFC 0.18124359959037378

NAIVE
Confusion Matrix:
[[925 77]
[67 931]]
DCF: 0.071965
DCF normalized: 0.143929
minDFC 0.1311283922171019

Prior: 0.90, Cost False Negative: 1.00, Cost False Positive: 1.00

QUI STAI DICENDO CHE A PRIORI È PIÚ ALTA LA PROB CHE SIA AUTENTICA L'IMPRONTA.

MVG
Confusion Matrix:
[[728 15]
[264 993]]
DCF: 0.040006
DCF normalized: 0.400058
minDFC 0.34230990783410137

TIED
Confusion Matrix:
[[666 15]
[326 993]]
DCF: 0.046256
DCF normalized: 0.462558
minDFC 0.4421082949308756

NAIVE
Confusion Matrix:
[[721 13]
[271 995]]
DCF: 0.038926
DCF normalized: 0.389257
minDFC 0.3509504608294931

Prior: 0.10, Cost False Negative: 1.00, Cost False Positive: 1.00

QUI CHE SIA FALSA

MVG
Confusion Matrix:
[[988 271]
[4 737]]
DCF: 0.030514
DCF normalized: 0.305140

minDFC 0.2629128264208909
TIED
Confusion Matrix:
[[983 327]
[9 681]]
DCF: 0.040606
DCF normalized: 0.406058
minDFC 0.36283922171018945
NAIVE
Confusion Matrix:
[[988 268]
[4 740]]
DCF: 0.030216
DCF normalized: 0.302163
minDFC 0.25696044546851
Prior: 0.50, Cost False Negative: 1.00, Cost False Positive: 9.00

QUI DA UN COSTO PIÙ ALTO SE CLASSIFICHI COME POSITIVO UN NEGATIVO. QUINDI DICE DI FARE ATTENZIONE E PIUTTOSTO A CLASSIFICARE COME NEGATIVO UN POSITIVO

MVG
Confusion Matrix:
[[988 271]
[4 737]]
DCF: 0.152570
DCF normalized: 0.305140
minDFC 0.262912826420891
TIED
Confusion Matrix:
[[983 327]
[9 681]]
DCF: 0.203029
DCF normalized: 0.406058
minDFC 0.36283922171018945
NAIVE
Confusion Matrix:
[[988 268]
[4 740]]
DCF: 0.151082
DCF normalized: 0.302163
minDFC 0.25696044546851
Prior: 0.50, Cost False Negative: 9.00, Cost False Positive: 1.00

QUI IL CONTRARIO

MVG
Confusion Matrix:
[[728 15]
[264 993]]
DCF: 0.200029
DCF normalized: 0.400058
minDFC 0.34230990783410137
TIED
Confusion Matrix:
[[666 15]
[326 993]]
DCF: 0.231279

DCF normalized: 0.462558
minDFC 0.44210829493087556
NAIVE
Confusion Matrix:
 [[721 13]
 [271 995]]
DCF: 0.194628
DCF normalized: 0.389257
minDFC 0.3509504608294931

Dai dati forniti, possiamo fare alcune osservazioni:

- Effetto della prior: Quando la prior è impostata a 0.90 o 0.10, il Decision Cost Function (DCF) tende ad essere più basso rispetto a quando la prior è impostata a 0.50. Questo suggerisce che il modello può avere prestazioni migliori quando la distribuzione delle classi è sbilanciata.
- Effetto del costo delle false negative e false positive: Quando il costo delle false negative è molto più alto del costo delle false positive (9.00 vs 1.00), il DCF tende ad essere più alto rispetto a quando i costi sono uguali. Questo suggerisce che il modello può avere difficoltà a minimizzare le false negative quando queste hanno un costo molto più alto.
- Confronto tra i modelli: Il modello MVG (Multivariate Gaussian) tende ad avere il DCF più basso tra i tre modelli in quasi tutti i casi. Questo suggerisce che il modello MVG potrebbe essere il più performante tra i tre per questo particolare set di dati e configurazione dei costi.
- Minimo DCF: Il minimo DCF fornisce una misura del miglior risultato possibile che si potrebbe ottenere variando la soglia di decisione. In tutti i casi, il minimo DCF è inferiore al DCF calcolato con la soglia di decisione attuale. Questo suggerisce che potrebbe essere possibile migliorare le prestazioni del modello ottimizzando la soglia di decisione.

SOLO SU QUESTI O TUTTI ?

[(0.1, 1, 1), (0.5, 1, 1), (0.9, 1, 1)]

? We now focus on the three applications, represented in terms of effective priors (i.e., with costs of errors equal to 1) given by $\tilde{\pi} = 0.1$, $\tilde{\pi} = 0.5$ and $\tilde{\pi} = 0.9$, respectively.

QUESTO SI RIFERISCE ALLA PROBABILITÀ A PRIORI CHE VIENE EFFETTIVAMENTE UTILIZZATA DOPO AVER PRESO IN CONSIDERAZIONE I DATI

1) For each application, compute the optimal Bayes decisions for the validation set for the MVG models and its variants, with and without PCA (try different values of m). Compute DCF (actual) and minimum DCF for the different models. Compare the models in terms of minimum DCF. Which models perform best? Are relative performance results consistent for the different applications? Now consider also actual DCFs. Are the models well calibrated (i.e., with a calibration loss in the range of few percents of the minimum DCF value) *for the given applications*? Are there models that are better calibrated than others for the considered applications?

ANALISI DEI RISULTATI

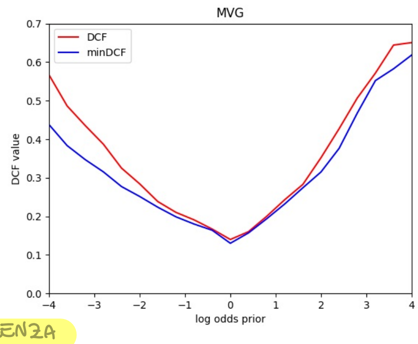
Consider now the PCA setup that gave the best results for the $\tilde{\pi} = 0.1$ configuration (this will be our main application). Compute the Bayes error plots for the MVG, Tied and Naive Bayes Gaussian classifiers. Compare the minimum DCF of the three models for different applications, and, for each model, plot minimum and actual DCF. Consider prior log odds in the range $(-4, +4)$. What do you observe? Are model rankings consistent across applications (minimum DCF)? Are models well-calibrated over the considered range?

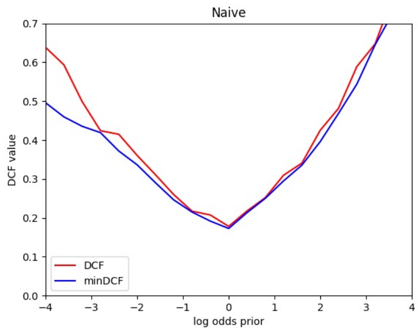
?

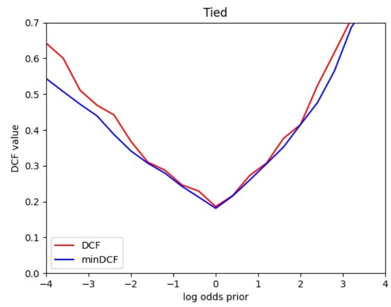
1) IL MODELLO PIÙ PERFORMANTE SEMBRA ESSERE MVG POICHÉ SEMBRA AVERE DCF PIÙ BASSO NELLA MAGGIOR PARTE DEI CASI.

- LA COERENZA TRA LE DIVERSE APPLICAZIONI PUÒ ESSERE CONFRONTATA GUARDANDO I DCF NORMALIZZATI SE TRA LE DIVERSE APPLICAZIONI È SIMILE ALLORA LE PRESTAZIONI SONO COERENTI.
- LA CALIBRAZIONE DEI MODELLI PUÒ ESSERE VALUTATO CONFRONTANDO IL DCF CON IL MINIMO DCF. È BEN CALIBRATO SE IL DCF È VICINO AL MINIMO DCF. TUTTI I MODELLI HANNO UN DCF MAGGIORE DEL MINIMO CHE SUGGERISCE CHE POTREBBERO NON ESSERE PERFETTAMENTE CALIBRATE.

PER DARE MIGLIORI GIUDIZI BISOGNEREBBE USARE ALTRE METRICHE COME L'AREA SOTTO LA CURVA ROC







COERENZA

SE PER OGNI MODELLO I VALORI DI DCF MINIMO PIÙ BASSI SONO COERENTI TRA LE DIVERSE APPLICAZIONI, ALLORA LE CLASSIFICHE DEI MODELLI SONO COERENTI.

CALIBRAZIONE

SE MIN E DCF SONO VICINI ALLORA VUOL DIRE CHE HANNO UNA BUONA CALIBRAZIONE INOLTRE SE AL VARIARE DEI VALORI RESTANO STABILI.

È BEN CALIBRATO POICHÉ I VALORI NON SI DISCOSTANO TROPPO

Apply PCA
PCA+MVG, m= 1
MVG 2-Class problem - Error rate: 9.250000%
Naive 2-Class problem - Error rate: 9.250000%
Tied 2-Class problem - Error rate: 9.350000%
Prior: 0.50, Cost False Negative: 1.00, Cost False Positive: 1.00
MVG
Confusion Matrix:
 [[900 93]
 [92 915]]
DCF: 0.092502
DCF normalized: 0.185004
minDFC 0.17690732206861237
TIED
Confusion Matrix:
 [[898 93]
 [94 915]]
DCF: 0.093510
DCF normalized: 0.187020
minDFC 0.17690732206861237
NAIVE
Confusion Matrix:
 [[900 93]
 [92 915]]
DCF: 0.092502
DCF normalized: 0.185004
minDFC 0.17690732206861237
Prior: 0.90, Cost False Negative: 1.00, Cost False Positive: 1.00
MVG
Confusion Matrix:
 [[660 16]
 [332 992]]
DCF: 0.047753
DCF normalized: 0.477535
minDFC 0.4341877880184332
TIED
Confusion Matrix:
 [[657 16]
 [335 992]]
DCF: 0.048056
DCF normalized: 0.480559
minDFC 0.4341877880184332
NAIVE
Confusion Matrix:
 [[660 16]
 [332 992]]
DCF: 0.047753
DCF normalized: 0.477535
minDFC 0.4341877880184332
Prior: 0.10, Cost False Negative: 1.00, Cost False Positive: 1.00

```
MVG
Confusion Matrix:
 [[984 327]
 [  8 681]]
DCF: 0.039699
DCF normalized: 0.396985
minDFC 0.3686475934459805
TIED
Confusion Matrix:
 [[983 323]
 [  9 685]]
DCF: 0.040209
DCF normalized: 0.402090
minDFC 0.3686475934459805
NAIVE
Confusion Matrix:
 [[984 327]
 [  8 681]]
DCF: 0.039699
DCF normalized: 0.396985
minDFC 0.3686475934459805
Prior: 0.50, Cost False Negative: 1.00, Cost False Positive: 9.00
MVG
Confusion Matrix:
 [[984 327]
 [  8 681]]
DCF: 0.198493
DCF normalized: 0.396985
minDFC 0.3686475934459805
TIED
Confusion Matrix:
 [[983 323]
 [  9 685]]
DCF: 0.201045
DCF normalized: 0.402090
minDFC 0.3686475934459805
NAIVE
Confusion Matrix:
 [[984 327]
 [  8 681]]
DCF: 0.198493
DCF normalized: 0.396985
minDFC 0.3686475934459805
Prior: 0.50, Cost False Negative: 9.00, Cost False Positive: 1.00
MVG
Confusion Matrix:
 [[660  16]
 [332 992]]
DCF: 0.238767
DCF normalized: 0.477535
```

minDFC 0.43418778801843316
TIED
Confusion Matrix:
 [[657 16]
 [335 992]]
DCF: 0.240279
DCF normalized: 0.480559
minDFC 0.43418778801843316
NAIVE
Confusion Matrix:
 [[660 16]
 [332 992]]
DCF: 0.238767
DCF normalized: 0.477535
minDFC 0.43418778801843316
PCA+MVG, m= 2
MVG 2-Class problem - Error rate: 8.800000%
Naive 2-Class problem - Error rate: 8.850000%
Tied 2-Class problem - Error rate: 9.250000%
Prior: 0.50, Cost False Negative: 1.00, Cost False Positive: 1.00
MVG
Confusion Matrix:
 [[905 89]
 [87 919]]
DCF: 0.087998
DCF normalized: 0.175995
minDFC 0.17308307731694828
TIED
Confusion Matrix:
 [[898 91]
 [94 917]]
DCF: 0.092518
DCF normalized: 0.185036
minDFC 0.1788594470046083
NAIVE
Confusion Matrix:
 [[906 91]
 [86 917]]
DCF: 0.088486
DCF normalized: 0.176971
minDFC 0.1710189452124936
Prior: 0.90, Cost False Negative: 1.00, Cost False Positive: 1.00
MVG
Confusion Matrix:
 [[685 15]
 [307 993]]
DCF: 0.044340
DCF normalized: 0.443404
minDFC 0.4383640552995392
TIED

Confusion Matrix:
[[659 16]
[333 992]]
DCF: 0.047854
DCF normalized: 0.478543
minDFC 0.4351958525345622
NAIVE
Confusion Matrix:
[[686 15]
[306 993]]
DCF: 0.044240
DCF normalized: 0.442396
minDFC 0.432315668202765
Prior: 0.10, Cost False Negative: 1.00, Cost False Positive: 1.00
MVG
Confusion Matrix:
[[985 327]
[7 681]]
DCF: 0.038791
DCF normalized: 0.387913
minDFC 0.35264656938044037
TIED
Confusion Matrix:
[[984 326]
[8 682]]
DCF: 0.039599
DCF normalized: 0.395993
minDFC 0.36298323092677925
NAIVE
Confusion Matrix:
[[985 326]
[7 682]]
DCF: 0.038692
DCF normalized: 0.386921
minDFC 0.35618279569892475
Prior: 0.50, Cost False Negative: 1.00, Cost False Positive: 9.00
MVG
Confusion Matrix:
[[985 327]
[7 681]]
DCF: 0.193956
DCF normalized: 0.387913
minDFC 0.35264656938044037
TIED
Confusion Matrix:
[[984 326]
[8 682]]
DCF: 0.197997
DCF normalized: 0.395993
minDFC 0.3629832309267793

NAIVE
Confusion Matrix:
 [[985 326]
 [7 682]]
DCF: 0.193460
DCF normalized: 0.386921
minDFC 0.35618279569892475
Prior: 0.50, Cost False Negative: 9.00, Cost False Positive: 1.00
MVG
Confusion Matrix:
 [[685 15]
 [307 993]]
DCF: 0.221702
DCF normalized: 0.443404
minDFC 0.43836405529953915
TIED
Confusion Matrix:
 [[659 16]
 [333 992]]
DCF: 0.239271
DCF normalized: 0.478543
minDFC 0.4351958525345622
NAIVE
Confusion Matrix:
 [[686 15]
 [306 993]]
DCF: 0.221198
DCF normalized: 0.442396
minDFC 0.432315668202765
PCA+MVG, m= 3
MVG 2-Class problem - Error rate: 8.800000%
Naive 2-Class problem - Error rate: 9.000000%
Tied 2-Class problem - Error rate: 9.250000%
Prior: 0.50, Cost False Negative: 1.00, Cost False Positive: 1.00
MVG
Confusion Matrix:
 [[911 95]
 [81 913]]
DCF: 0.087950
DCF normalized: 0.175899
minDFC 0.17343509984639016
TIED
Confusion Matrix:
 [[900 93]
 [92 915]]
DCF: 0.092502
DCF normalized: 0.185004
minDFC 0.18301971326164873
NAIVE
Confusion Matrix:

[[907 95]
 [85 913]]
DCF: 0.089966
DCF normalized: 0.179932
minDFC 0.17461917562724014
Prior: 0.90, Cost False Negative: 1.00, Cost False Positive: 1.00
MVG
Confusion Matrix:
 [[696 19]
 [296 989]]
DCF: 0.046803
DCF normalized: 0.468030
minDFC 0.4392281105990784
TIED
Confusion Matrix:
 [[672 15]
 [320 993]]
DCF: 0.045651
DCF normalized: 0.456509
minDFC 0.43418778801843332
NAIVE
Confusion Matrix:
 [[696 18]
 [296 990]]
DCF: 0.045910
DCF normalized: 0.459101
minDFC 0.43433179723502313
Prior: 0.10, Cost False Negative: 1.00, Cost False Positive: 1.00
MVG
Confusion Matrix:
 [[985 327]
 [7 681]]
DCF: 0.038791
DCF normalized: 0.387913
minDFC 0.35632680491551455
TIED
Confusion Matrix:
 [[982 320]
 [10 688]]
DCF: 0.040819
DCF normalized: 0.408186
minDFC 0.3680875576036866
NAIVE
Confusion Matrix:
 [[984 325]
 [8 683]]
DCF: 0.039500
DCF normalized: 0.395001
minDFC 0.36453533026113677
Prior: 0.50, Cost False Negative: 1.00, Cost False Positive: 9.00

MVG
Confusion Matrix:
 [[985 327]
 [7 681]]
DCF: 0.193956
DCF normalized: 0.387913
minDFC 0.3563268049155146
TIED
Confusion Matrix:
 [[982 320]
 [10 688]]
DCF: 0.204093
DCF normalized: 0.408186
minDFC 0.3680875576036866
NAIVE
Confusion Matrix:
 [[984 325]
 [8 683]]
DCF: 0.197501
DCF normalized: 0.395001
minDFC 0.3645353302611367
Prior: 0.50, Cost False Negative: 9.00, Cost False Positive: 1.00
MVG
Confusion Matrix:
 [[696 19]
 [296 989]]
DCF: 0.234015
DCF normalized: 0.468030
minDFC 0.4392281105990783
TIED
Confusion Matrix:
 [[672 15]
 [320 993]]
DCF: 0.228255
DCF normalized: 0.456509
minDFC 0.43418778801843316
NAIVE
Confusion Matrix:
 [[696 18]
 [296 990]]
DCF: 0.229551
DCF normalized: 0.459101
minDFC 0.434331797235023
PCA+MVG, m= 4
MVG 2-Class problem - Error rate: 8.050000%
Naive 2-Class problem - Error rate: 8.850000%
Tied 2-Class problem - Error rate: 9.250000%
Prior: 0.50, Cost False Negative: 1.00, Cost False Positive: 1.00
MVG
Confusion Matrix:

[[916 85]
 [76 923]]
DCF: 0.080469
DCF normalized: 0.160938
minDFC 0.15372183819764465
TIED
Confusion Matrix:
 [[899 92]
 [93 916]]
DCF: 0.092510
DCF normalized: 0.185020
minDFC 0.18210765488991296
NAIVE
Confusion Matrix:
 [[906 91]
 [86 917]]
DCF: 0.088486
DCF normalized: 0.176971
minDFC 0.17167498719918073
Prior: 0.90, Cost False Negative: 1.00, Cost False Positive: 1.00
MVG
Confusion Matrix:
 [[713 20]
 [279 988]]
DCF: 0.045982
DCF normalized: 0.459821
minDFC 0.4150345622119816
TIED
Confusion Matrix:
 [[667 15]
 [325 993]]
DCF: 0.046155
DCF normalized: 0.461550
minDFC 0.44412442396313373
NAIVE
Confusion Matrix:
 [[701 19]
 [291 989]]
DCF: 0.046299
DCF normalized: 0.462990
minDFC 0.431307603686636
Prior: 0.10, Cost False Negative: 1.00, Cost False Positive: 1.00
MVG
Confusion Matrix:
 [[987 310]
 [5 698]]
DCF: 0.035290
DCF normalized: 0.352903
minDFC 0.3011872759856631
TIED

Confusion Matrix:
 [[983 324]
 [9 684]]
DCF: 0.040308
DCF normalized: 0.403082
minDFC 0.3609991039426523
NAIVE
Confusion Matrix:
 [[984 327]
 [8 681]]
DCF: 0.039699
DCF normalized: 0.396985
minDFC 0.3614151305683564
Prior: 0.50, Cost False Negative: 1.00, Cost False Positive: 9.00
MVG
Confusion Matrix:
 [[987 310]
 [5 698]]
DCF: 0.176451
DCF normalized: 0.352903
minDFC 0.3011872759856631
TIED
Confusion Matrix:
 [[983 324]
 [9 684]]
DCF: 0.201541
DCF normalized: 0.403082
minDFC 0.36099910394265233
NAIVE
Confusion Matrix:
 [[984 327]
 [8 681]]
DCF: 0.198493
DCF normalized: 0.396985
minDFC 0.3614151305683564
Prior: 0.50, Cost False Negative: 9.00, Cost False Positive: 1.00
MVG
Confusion Matrix:
 [[713 20]
 [279 988]]
DCF: 0.229911
DCF normalized: 0.459821
minDFC 0.41503456221198154
TIED
Confusion Matrix:
 [[667 15]
 [325 993]]
DCF: 0.230775
DCF normalized: 0.461550
minDFC 0.4441244239631337

NAIVE
Confusion Matrix:
 [[701 19]
 [291 989]]
DCF: 0.231495
DCF normalized: 0.462990
minDFC 0.43130760368663595
PCA+MVG, m= 5
MVG 2-Class problem - Error rate: 7.100000%
Naive 2-Class problem - Error rate: 8.750000%
Tied 2-Class problem - Error rate: 9.300000%
Prior: 0.50, Cost False Negative: 1.00, Cost False Positive: 1.00
MVG
Confusion Matrix:
 [[927 77]
 [65 931]]
DCF: 0.070957
DCF normalized: 0.141913
minDFC 0.13314452124935997
TIED
Confusion Matrix:
 [[897 91]
 [95 917]]
DCF: 0.093022
DCF normalized: 0.186044
minDFC 0.1811635944700461
NAIVE
Confusion Matrix:
 [[906 89]
 [86 919]]
DCF: 0.087494
DCF normalized: 0.174987
minDFC 0.1736911162314388
Prior: 0.90, Cost False Negative: 1.00, Cost False Positive: 1.00
MVG
Confusion Matrix:
 [[730 15]
 [262 993]]
DCF: 0.039804
DCF normalized: 0.398041
minDFC 0.35123847926267276
TIED
Confusion Matrix:
 [[666 15]
 [326 993]]
DCF: 0.046256
DCF normalized: 0.462558
minDFC 0.44513248847926273
NAIVE
Confusion Matrix:

[[698 19]
[294 989]]
DCF: 0.046601
DCF normalized: 0.466014
minDFC 0.4340437788018433
Prior: 0.10, Cost False Negative: 1.00, Cost False Positive: 1.00
MVG
Confusion Matrix:
[[988 270]
[4 738]]
DCF: 0.030415
DCF normalized: 0.304147
minDFC 0.27382552483358935
TIED
Confusion Matrix:
[[983 326]
[9 682]]
DCF: 0.040507
DCF normalized: 0.405066
minDFC 0.36482334869431643
NAIVE
Confusion Matrix:
[[984 323]
[8 685]]
DCF: 0.039302
DCF normalized: 0.393017
minDFC 0.3544706861239119
Prior: 0.50, Cost False Negative: 1.00, Cost False Positive: 9.00
MVG
Confusion Matrix:
[[988 270]
[4 738]]
DCF: 0.152074
DCF normalized: 0.304147
minDFC 0.27382552483358935
TIED
Confusion Matrix:
[[983 326]
[9 682]]
DCF: 0.202533
DCF normalized: 0.405066
minDFC 0.36482334869431643
NAIVE
Confusion Matrix:
[[984 323]
[8 685]]
DCF: 0.196509
DCF normalized: 0.393017
minDFC 0.3544706861239119
Prior: 0.50, Cost False Negative: 9.00, Cost False Positive: 1.00

```
MVG
Confusion Matrix:
 [[730  15]
 [262 993]]
DCF: 0.199021
DCF normalized: 0.398041
minDFC 0.3512384792626728
TIED
Confusion Matrix:
 [[666  15]
 [326 993]]
DCF: 0.231279
DCF normalized: 0.462558
minDFC 0.4451324884792627
NAIVE
Confusion Matrix:
 [[698  19]
 [294 989]]
DCF: 0.233007
DCF normalized: 0.466014
minDFC 0.4340437788018433
PCA+MVG, m= 6
MVG 2-Class problem - Error rate: 7.000000%
Naive 2-Class problem - Error rate: 8.900000%
Tied 2-Class problem - Error rate: 9.300000%
Prior: 0.50, Cost False Negative: 1.00, Cost False Positive: 1.00
MVG
Confusion Matrix:
 [[927  75]
 [ 65 933]]
DCF: 0.069964
DCF normalized: 0.139929
minDFC 0.13016833077316947
TIED
Confusion Matrix:
 [[898  92]
 [ 94 916]]
DCF: 0.093014
DCF normalized: 0.186028
minDFC 0.18124359959037378
NAIVE
Confusion Matrix:
 [[904  90]
 [ 88 918]]
DCF: 0.088998
DCF normalized: 0.177995
minDFC 0.1726990527393753
Prior: 0.90, Cost False Negative: 1.00, Cost False Positive: 1.00
MVG
Confusion Matrix:
```

[[728 15]
 [264 993]]
DCF: 0.040006
DCF normalized: 0.400058
minDFC 0.34230990783410137
TIED
Confusion Matrix:
 [[666 15]
 [326 993]]
DCF: 0.046256
DCF normalized: 0.462558
minDFC 0.4421082949308756
NAIVE
Confusion Matrix:
 [[695 17]
 [297 991]]
DCF: 0.045118
DCF normalized: 0.451181
minDFC 0.43591589861751157
Prior: 0.10, Cost False Negative: 1.00, Cost False Positive: 1.00
MVG
Confusion Matrix:
 [[988 271]
 [4 737]]
DCF: 0.030514
DCF normalized: 0.305140
minDFC 0.2629128264208909
TIED
Confusion Matrix:
 [[983 327]
 [9 681]]
DCF: 0.040606
DCF normalized: 0.406058
minDFC 0.36283922171018945
NAIVE
Confusion Matrix:
 [[984 322]
 [8 686]]
DCF: 0.039203
DCF normalized: 0.392025
minDFC 0.3534786226318484
Prior: 0.50, Cost False Negative: 1.00, Cost False Positive: 9.00
MVG
Confusion Matrix:
 [[988 271]
 [4 737]]
DCF: 0.152570
DCF normalized: 0.305140
minDFC 0.262912826420891
TIED

```
Confusion Matrix:
 [[983 327]
 [  9 681]]
DCF: 0.203029
DCF normalized: 0.406058
minDFC 0.36283922171018945
NAIVE
Confusion Matrix:
 [[984 322]
 [  8 686]]
DCF: 0.196013
DCF normalized: 0.392025
minDFC 0.3534786226318484
Prior: 0.50, Cost False Negative: 9.00, Cost False Positive: 1.00
MVG
Confusion Matrix:
 [[728  15]
 [264 993]]
DCF: 0.200029
DCF normalized: 0.400058
minDFC 0.34230990783410137
TIED
Confusion Matrix:
 [[666  15]
 [326 993]]
DCF: 0.231279
DCF normalized: 0.462558
minDFC 0.44210829493087556
NAIVE
Confusion Matrix:
 [[695  17]
 [297 991]]
DCF: 0.225590
DCF normalized: 0.451181
minDFC 0.4359158986175115
```