Score calibration and fusion

Sandro Cumani

sandro.cumani@polito.it

Politecnico di Torino

Score calibration

We have seen that, in many cases, classifiers output scores that have a weak, or no probabilistic interpretation, i.e. they do not represent LLRs for the application (evaluation) population

- SVMs — scores have no probabilistic interpretation
- Regularized logistic regression — strong regularization adversely influence the probabilistic interpretation of the score
- Overfitting or distribution mismatch may affect even generative models

Decisions made with the classifier scores may be sub-optimal (miscalibration)

Score calibration (binary tasks)

For binary problems we have seen that decision rules take the form of a comparison of the classifier score with a threshold

To reduce miscalibration we can adopt different strategies

We can use a validation set to find a (close-to) optimal threshold for a given application

- We need to know the target application when estimating the threshold

Alternatively, we can look for functions that transform the classifier scores into approximately well-calibrated LLRs, so that optimal decisions can be obtained for different applications

- Application independent (to some degree)
- Prior knowledge of the application can still help for some models, but a rough estimate of possible applications is often sufficient

Score calibration (binary tasks)

Let's consider the second approach

We want to compute a transformation function f that maps a classifier score s into a well-calibrated score $s_{cal} = f(s)$

We consider monotone functions f, as to preserve the fact that higher non-calibrated scores should favor class $\mathcal{H}_T$ and lower non-calibrated scores should favor class $\mathcal{H}_F$

- Isotonic regression
- Prior-weighted logistic regression (with affine scoring function)
- Generative score models (with constraints)

Score calibration (binary tasks)

Isotonic regression

- Non-linear, monotonic transformation that provides optimal calibration for the data it's trained on
- Piece-wise non-linear, may require some sort of interpolation for unseen scores
- Does not allow extrapolating outside of training scores range
- Expensive to evaluate when the calibration training set is large

Score calibration (binary tasks)

Score models

- Require assumptions on the calibration transformation (e.g. affine mapping) or on the distribution of class scores
- Estimated over a training set, but allow for extrapolation outside of training score ranges
- Typically, fast to evaluate
- May provide good fit only for a small range of operating points

Example: Prior-weighted logistic regression

Score calibration (binary tasks)

Calibration: prior-weighted logistic regression

We consider the non-calibrated scores as 1-D feature vectors

We assume an affine mapping from non-calibrated scores to calibrated scores

$$f(s) = \alpha s + \gamma$$

Since $f(s)$ should produce well-calibrated LLRs, $f(s)$ can be interpreted as the log-likelihood ratio for the two class hypotheses

$$f(s) = \log \frac{f_{S|C}(s|\mathcal{H}_T)}{f_{S|C}(s|\mathcal{H}_F)} = \alpha s + \gamma$$

The class posterior probabilities for prior $\tilde{\pi}$ correspond to

$$\log \frac{P(C = \mathcal{H}_T|s)}{P(C = \mathcal{H}_F|s)} = \alpha s + \gamma + \log \frac{\tilde{\pi}}{1 - \tilde{\pi}} = \alpha s + \beta$$

Score calibration (binary tasks)

We can employ the (non-regularized) prior-weighted logistic regression model to learn the model parameters α, β from a set of training scores (we will refer to the set of samples employed to estimate the calibration parameters as *calibration training set* in the following)

In practice, we are treating socres as if they were 1-D feature vectors

As for model training and validation set, also the calibration set should be an independent dataset that does not overlap with neither the model training nor the validation set (we will see later how to effectively split the data)

Score calibration (binary tasks)

The calibration transformation corresponds to the transformation of a prior-weighted logistic regression model

Once we have estimated α and β, the calibrated score $f(s)$ can be computed as

$$f(s) = \alpha s + \gamma = \alpha s + \beta - \log \frac{\tilde{\pi}}{1 - \tilde{\pi}}$$

Note that we have to specify a prior $\tilde{\pi}$, and we are still effectively optimizing the calibration for a specific application $\tilde{\pi}$. However, often this approach will provide good calibration for a wider range of different applications similar to the target one

We can also modify the prior-weighted logistic regression objective to compute directly α and γ:

$$R(\alpha, \gamma) = \sum_i w_i \log \left(1 + e^{-z_i\left(\alpha s + \gamma + \log \frac{\tilde{\pi}}{1-\tilde{\pi}}\right)}\right) , \quad w_i = \begin{cases} \tilde{\pi}/n_T & \text{if } z_i = +1 \\ (1-\tilde{\pi})/n_F & \text{if } z_i = -1 \end{cases}$$

Score calibration (binary tasks)

Alternatively to logistic regression, we can employ generative models to estimate calibrated LLRs

We model the class-conditional *score* distribution for the two classes

$$S|\mathcal{H}_T \ , \quad S|\mathcal{H}_F$$

For example, we can employ a Gaussian model to estimate the two densities

$$f_{S|\mathcal{H}_T}(s) = \mathcal{N}(s|\mu_T, v_F) \ , \quad f_{S|\mathcal{H}_F}(s) = \mathcal{N}(s|\mu_T, v_F)$$

and the calibration transformation is given by the LLR for the score model

$$f_{cal}(s) = \log \frac{f_{S|\mathcal{H}_T}(s)}{f_{S|\mathcal{H}_F}(s)}$$

Tied models $v_T = v_F = v$ are usually employed, as they result in monotonic transformations

Score calibration

In all cases, we need a calibration set to estimate the transformation

The calibration set must differ from the validation / evaluation sets (otherwise we would get biased results)

Typically, two scenarios:

1. Miscalibration due to non-probabilistic scores, or to overfitting or underfitting models, but evaluation and training populations are similar: the calibration set can be extracted from the training set material

   In this case, calibration allows recovering a probabilistic interpretation of the scores, and, since we are training on 1-D data, the risk of overfitting is drastically reduced (and typically we don't need to add regularization to the calibration model objective function)

Score calibration

Typically, two scenarios:

2. Miscalibration due to mismatch between training and the application / evaluation population: the calibration set (and in this case the calibration validation set as well, if we want to assess the goodness of the calibration) should mimic the application population

   We need to collect data similar to the application population

   The amount of required matching data, however, is usually small compared to the amount required to train a complete classifier

   Some models (e.g. Gaussian) can be extended to exploit unlabeled samples (unsupervised or semi-supervised training), which are less expensive to acquire

Score calibration

A note on projects: our use-case falls in the first scenario

The calibration training and validation sets can be taken from the training set we employed up to now (the evaluation set, that will be disclosed at the end of the course, *cannot* be used to estimate any part of the model)

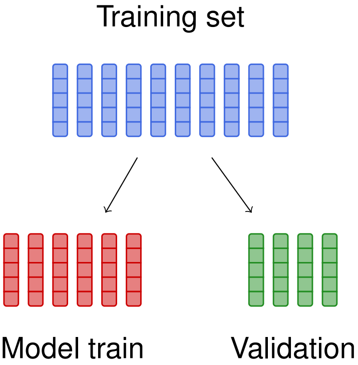
Score calibration

The approach we followed up to now would require that we split the training material in 3 parts:

- Model training
- Calibration training set (samples that are scored with the trained classifier and are used to compute calibration parameters)
- (Calibration) validation set (samples that are scored with the trained model and whose scores are calibrated with the calibration model, used to evaluate the performance of the complete system, including the calibration model)

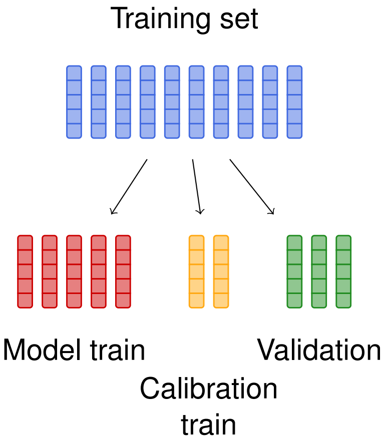
Score calibration

Current set-up (without calibration dataset):



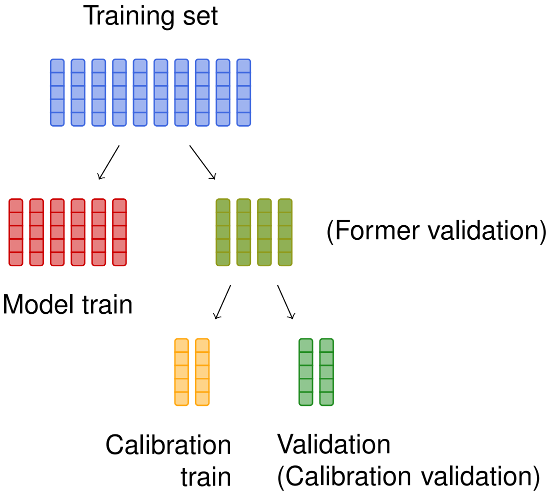
Score calibration

Train - calibration - validation:



Score calibration

Splits can be performed in several ways, for simplicity, we assume that our calibration training and validation sets are extracted from our former validation set (this allows us to re-use models we already trained)



Score calibration

Issue: the calibration and validation sets become both smaller

If we enlarge the calibration set, our evaluation metrics computed over the validation set become less reliable

If we reduce the calibration set, our calibration model becomes less reliable

In general, all three partitions would benefit from more data (with the calibration partition being usually the least critical)

A possible approach to address the limited amount of data is to resort to K-fold cross-validation

K-fold

Single-fold partitions the dataset in separate sets

K-fold extends the approach by performing K times the split, in a way that allows us to exploit all data both for model / calibration training and evaluation

We start considering a simple set-up, where we need two partitions:

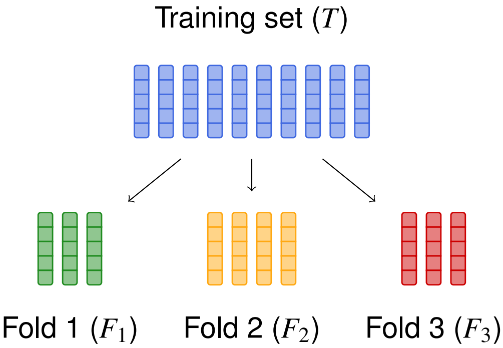
- these could be model training and validation sets extracted from a training dataset
- these could be the calibration and actual validation sets extracted from our former validation set

In both cases, we would like to use the data to train some model (classifier or calibration model) and to evaluate its performance

K-fold

K-fold: split the dataset in K partitions (usually with similar amounts of samples, keeping the same class ratios as the original set)

With K = 3:

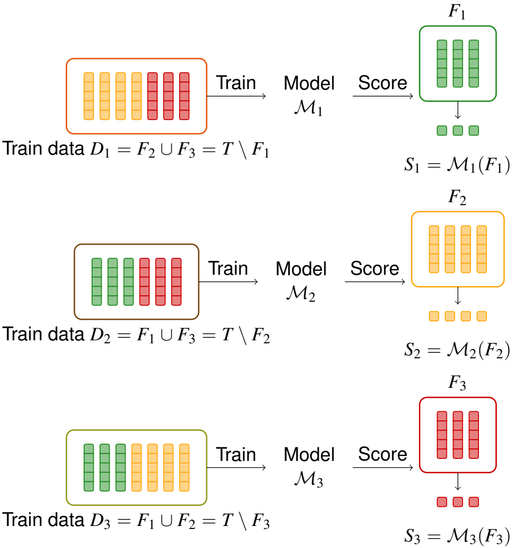


Training set (T)

Fold 1 (F_1)   Fold 2 (F_2)   Fold 3 (F_3)

K-fold

Iteratively train using K-1 folds, and scoring the left-out fold:



F_1

Train data $D_1 = F_2 \cup F_3 = T \setminus F_1$ $\xrightarrow{\text{Train}}$ Model $\mathcal{M}_1$ $\xrightarrow{\text{Score}}$ F_1

$S_1 = \mathcal{M}_1(F_1)$

F_2

Train data $D_2 = F_1 \cup F_3 = T \setminus F_2$ $\xrightarrow{\text{Train}}$ Model $\mathcal{M}_2$ $\xrightarrow{\text{Score}}$ F_2

$S_2 = \mathcal{M}_2(F_2)$

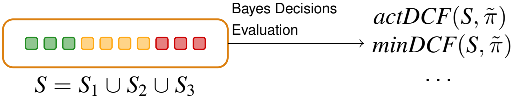
F_3

Train data $D_3 = F_1 \cup F_2 = T \setminus F_3$ $\xrightarrow{\text{Train}}$ Model $\mathcal{M}_3$ $\xrightarrow{\text{Score}}$ F_3

$S_3 = \mathcal{M}_3(F_3)$

K-fold

Pool together the scores of the different folds and evaluate the desired metric



Bayes Decisions Evaluation

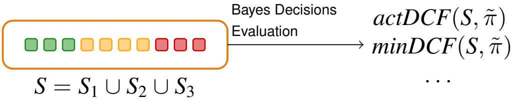
$$S = S_1 \cup S_2 \cup S_3 \xrightarrow{} \begin{array}{c} actDCF(S, \tilde{\pi}) \\ minDCF(S, \tilde{\pi}) \\ \cdots \end{array}$$

Use the chosen metric to perform model selection / hyperparameter tuning for each model

NOTE (1): all the K models $\mathcal{M}_1 \ldots \mathcal{M}_K$ *must* be trained with the same set-up (i.e., same pre-processing, same values for the hyperparameters, ...)

K-fold

Pool together the scores of the different folds and evaluate the desired metric



Bayes Decisions Evaluation

$$S = S_1 \cup S_2 \cup S_3 \xrightarrow{\text{Bayes Decisions Evaluation}} \begin{array}{c} actDCF(S, \tilde{\pi}) \\ minDCF(S, \tilde{\pi}) \\ \cdots \end{array}$$

Use the chosen metric to perform model selection / hyperparameter tuning for each model

NOTE (2): For some metrics, including minDCF, it’s essential that we compute the metric over the *pooled* scores (i.e., over the set obtained by merging the scores of all folds)

- Averaging the metric computed over each fold would lead to *biased* results (and is thus wrong)
- We need to use the *same* threshold for *all* scores

K-fold

Once we have selected the approach (model selection) and tuned the model parameters, we still need to choose which model to use in practice

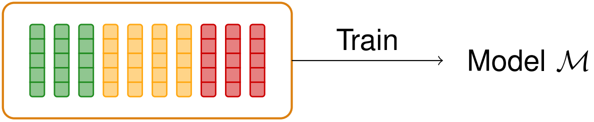
We have trained K models, all of them using a subset of the data

We cannot chose any of them based on their performance on the corresponding validation fold - fold data are different, and the comparison would then make little sense

Moreover, in many cases increasing the training set size is beneficial, but each model was trained using only approximately a fraction $\frac{K-1}{K}$ of the training set

K-fold

Solution: we train one more model over the *whole* training set, using the method we selected and the hyperparameters we estimated in the previous step



Train

Model $\mathcal{M}$

$$T = F_1 \cup F_2 \cup F_3$$

The model $\mathcal{M}$ will then be used to compute the scores for the application data

K-fold

Note that model selection and hyperparameter tuning has been performed using models that are different from $\mathcal{M}$

The selected values may not generalize well for the final model (e.g., the optimal regularization parameters λ or C of logistic regression or SVM may depend on the dataset size)

Furthermore, each model $\mathcal{M}_i$ may be affacted by different levels of miscalibration

Ideally, to minimize these issue we want each model $\mathcal{M}_i$ to be as similar as possible to the final model $\mathcal{M}$

K-fold

Leave-one-out (LOO): K-fold with K equal to the *number of training samples N*

Each time, we remove a single sample from training, and we score it with a model trained over all remaining samples

We train each model with $N - 1$ samples

- The models will be usually very similar
- The *pooled* validation set consists of N samples obtained from similar models

K-fold

Unfortunately, leave-one-out may require training an excessively large number of models (we need to train N times a model over $N-1$ samples

We choose K as large as we can afford

- Large values of K: more robust analysis, more time required
- Small values of K: less robust analysis, but faster

K-fold

We can extend the approach to deal with our three-set partitioning using a two-step approach

First step: apply K-fold to train the classifier

- Train K classifiers, each without fold k (model $\mathcal{R}_k$)
- Score each fold k with model $\mathcal{R}_k$
- Train a classifier $\mathcal{R}_\mathcal{F}$ over the whole training set (we will need this later)
- Pool the scores of each fold to obtain a score set, that we will use as calibration set

K-fold

Second step: apply K-fold over the calibration scores, using a different randomized shuffle

- Train K calibration models $\mathcal{C}_k$
- Train a calibration model over all pooled scores $\mathcal{C}_{\mathcal{F}}$ (we will need this later)
- Calibrate the scores of each fold k with model $\mathcal{C}_k$
- Pool the calibrated scores and evaluate the model performance over the pooled scores (to choose the best configuration)

Chose the classification system according to pooled scores results

Classify the application samples: apply classification model $\mathcal{R}_{\mathcal{F}}$ for the chosen system followed by its calibration model $\mathcal{C}_{\mathcal{F}}$

K-fold

For the course project you can use a simple split for model training

In the last laboratory you will be asked to apply K-fold for *the calibration part only*

You will have to apply the K-fold procedure for the estimation of a calibration model from the scores of the validation set you used up to now

Take the validation scores that you computed, split them in K-folds, train K calibration models, score the K folds with the calibration models, pool together the scores and evaluate the effectiveness of the calibration transformation

Score-level fusion

In many cases we may have competing models that achieve similar performance

In some cases, even when performance is significantly different, different classifiers may be able to extract different information from the feature vectors

Can we combine these models to further boost our classification performance for a given test sample?

Score-level fusion

Simple strategies exploit majority-voting schemes. Whenever we need to classify a sample $\boldsymbol{x}$:

- Compute the prediction of each of the m considered classifier $p_1 \dots p_m$ for sample $\boldsymbol{x}$
- Assign the label that is selected the most by the different models

While sometimes effective, this approach has some limitations:

- Requires tie-breaking rules (which becomes critical when we have only 2 recognizers)
- Does not account for the strength that each classifier gives towards each labeling hypothesis

Score-level fusion

Consider 3 classifiers that provide, for a sample $\boldsymbol{x}$, the scores $s_1 = 10.0, s_2 = -0.1, s_3 = -0.1$ (binary task, well calibrated scores)

- If our threshold is 0, then majority voting would label the sample as class $\mathcal{H}_F$
- The scores tell us that two systems are very uncertain, with a slight bias towards class $\mathcal{H}_F$, but the first one is very confident in favor of class $\mathcal{H}_T$

Score-level fusion

If our models provide scores as output, then we can try combining the scores for a given sample

For example, we may try computing the average or the sum of the scores

- If our models provide LLRs, and the class-conditional likelihoods provided by the systems were independent, then the sum would be correct (cfr. naive Bayes models)— however, the different systems usually employ the same or strongly correlated features
- If our models are strongly correlated the sum becomes incorrect, as some systems are providing only partial (or even no) additional information

Score-level fusion

We can consider weighting the contribution of the different systems. Given scores $s_1 \dots s_m$ for a test sample $\boldsymbol{x}$, we can compute a "fused" score as

$$s_{fused} = \alpha_1 s_1 + \alpha_2 s_2 + \alpha_3 s_3 + \dots + \alpha_m s_m + \gamma$$

We have the problem of estimating the weights α_i and the bias term γ

If we *stack the scores* and the weights as

$$\boldsymbol{s} = \begin{bmatrix} s_1 \\ \vdots \\ s_m \end{bmatrix} \quad , \quad \boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_m \end{bmatrix}$$

the fused score can be represented as

$$s_{fused} = \boldsymbol{\alpha}^T \boldsymbol{s} + \gamma$$

Score-level fusion

The output score s_{fused} is obtained as an affine transformation of the score "feature" vector $\boldsymbol{s}$

We can then employ a method like logistic regression to estimate the weights $\boldsymbol{\alpha}$ and the bias term γ (if we want a LLR-like fused scores we need to compensate the logistic regression model parameters for the training prior log-odds as in the calibration case to obtain γ)

This is similar to calibration (and typically provides calibrated scores as output), but rather than 1-D samples we now have m-dimensional vectors, containing the scores of m classifiers for each sample

We can exploit the calibration partition of our set to estimate the score fusion weights

If we had a single system, we would obtain exactly the calibration procedure we analyzed earlier

Multiclass calibration and fusion

For multiclass problems the optimal decisions cannot be represented anymore as a comparison of a score with a threshold

In this case, we have seen that it's more complex to separate the contributions to the cost due to poor discrimination and poor calibration

Calibration methods can be extended to calibrate also scores of multiclass recognizers, so that they can be interpreted as class-conditional log-likelihoods that can then be combined with application priors and costs to obtain optimal decisions

Typically, multiclass logistic regression models are employed to estimate calibration or fusion weights and biases for the different classes and classifiers