Classifier Calibration

Peter Flach Miquel Perello Nieto

Taking inspiration from forecasting

Weather forecasters

- Weather forecasters started thinking about calibration a long time ago (Brier 1950).
  - A forecast `70% chance of rain` should be followed by rain 70% of the time.
- This is immediately applicable to binary classification:
  - A prediction `70% chance of spam` should be spam 70% of the time.
- and to multi-class classification:
  - A prediction `70% chance of setosa, 10% chance of versicolor and 20% chance of virginica` should be setosa/versicolor/virginica 70/10/20% of the time.
- In general:
  - A predicted probability (vector) should match empirical (observed) probabilities.

Q: What does `X% of the time` mean?

It means that we expect the occurrence of an event to happen "X%" of the time.

Forecasting example

Let's consider a small toy example:

- Two predictions of `10% chance of rain` were both followed by `no rain`.
- Two predictions of `40% chance of rain` were once followed by `no rain`, and once by `rain`.

- Three predictions of `70% chance of rain` were once followed by `no rain`, and twice by `rain`.
- One prediction of `90% chance of rain` was followed by `rain`.

Q: Is this forecaster well-calibrated?

The evaluation of calibration requires a large number of samples to make a statement. However, in this toy example we can assume that a 10% discrepancy is acceptable, and that the number of samples is sufficient.

Over- and under-estimates

	$\hat{p}$	y
0	0.1	0
1	0.1	0
2	0.4	0
3	0.4	1
4	0.7	0
5	0.7	1
6	0.7	1
7	0.9	1

This forecaster is doing a pretty decent job:

- `10% chance of rain` was a slight over-estimate ($\bar{y} = 0/2 = 0\%$);
- `40% chance of rain` was a slight under-estimate ($\bar{y} = 1/2 = 50\%$);
- `70% chance of rain` was a slight over-estimate ($\bar{y} = 2/3 = 67\%$);
- `90% chance of rain` was a slight under-estimate ($\bar{y} = 1/1 = 100\%$).

Visualising forecasts: the reliability diagram

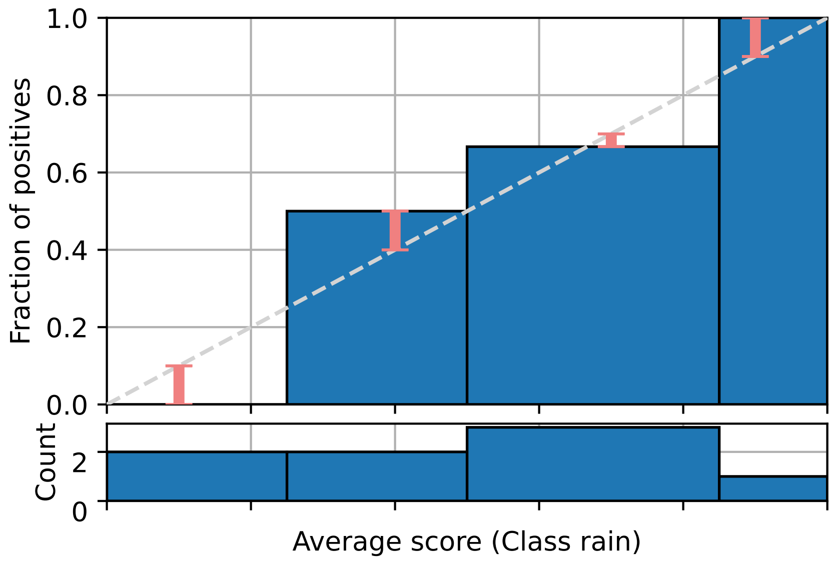
	$\hat{p}$	y
0	0.1	0
1	0.1	0

	$\hat{p}$	y
2	0.4	0
3	0.4	1
4	0.7	0
5	0.7	1
6	0.7	1
7	0.9	1

```python
import numpy as np
import matplotlib.pyplot as plt

from pycalib.visualisations import plot_reliability_diagram

labels = np.array([0, 0, 0, 1, 0, 1, 1, 1])
scores = np.array([0.1, 0.1 ,0.4, 0.4,0.7, 0.7, 0.7, 0.9])
bins = [0, 0.25, 0.5, 0.85, 1.0]
fig = plt.figure(figsize=(5, 4))
fig = plot_reliability_diagram(labels, np.vstack([1 - scores, scores]).T,
                               class_names=['not 1', 'rain'], bins=bins,
                               fig=fig, show_gaps=True,
                               show_bars=True)
```



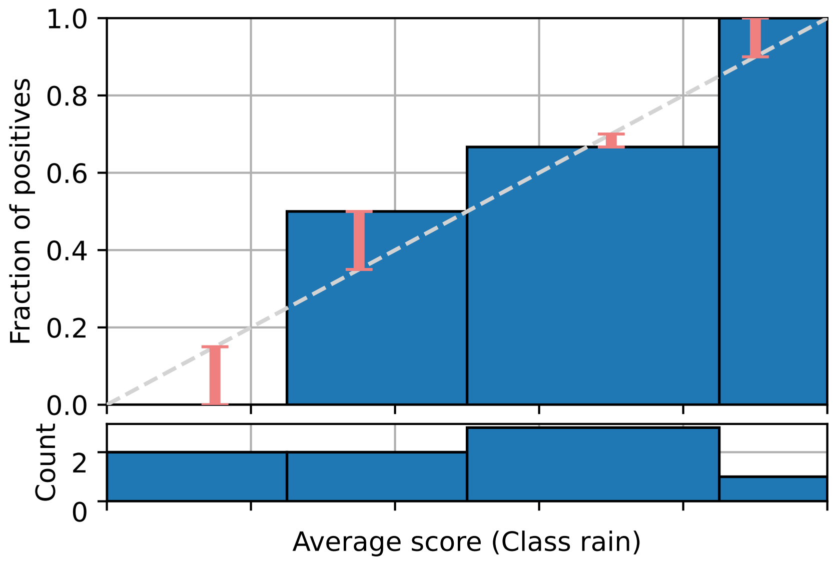
Changing the numbers slightly

	$\hat{p}$	y
0	0.1	0
1	0.2	0
2	0.3	0
3	0.4	1
4	0.6	0
5	0.7	1
6	0.8	1
7	0.9	1

```python
import numpy as np
import matplotlib.pyplot as plt

from pycalib.visualisations import plot_reliability_diagram

labels = np.array([0, 0, 0, 1, 0, 1, 1, 1])
scores = np.array([0.1, 0.2 ,0.3, 0.4, 0.6, 0.7, 0.8, 0.9])
bins = [0, 0.25, 0.5, 0.85, 1.0]
fig = plt.figure(figsize=(5, 4))
fig = plot_reliability_diagram(labels, np.vstack([1 - scores, scores]).T,
                               class_names=['not 1', 'rain'], bins=bins,
                               fig=fig, show_gaps=True,
                               show_bars=True)
```



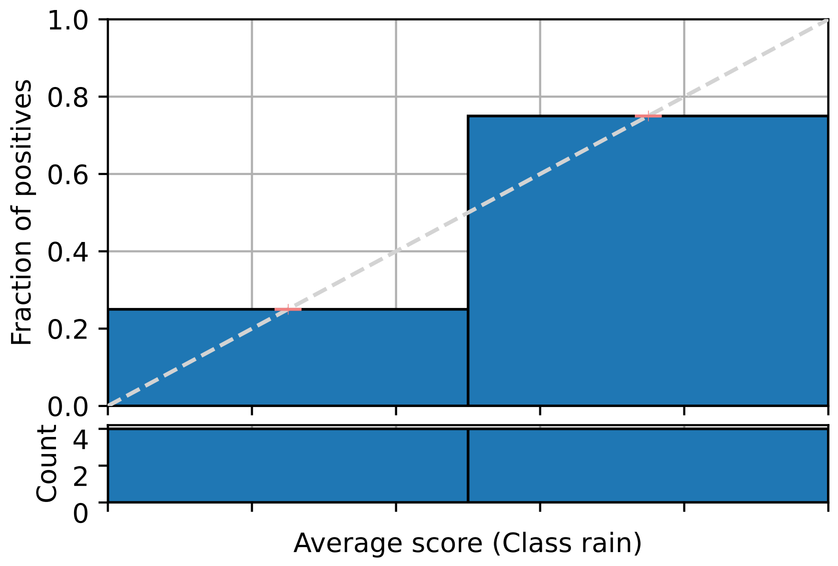
Or should we group the forecasts differently?

	$\hat{p}$	y
0	0.1	0
1	0.2	0
2	0.3	0
3	0.4	1
4	0.6	0
5	0.7	1
6	0.8	1
7	0.9	1

```python
import numpy as np
import matplotlib.pyplot as plt

from pycalib.visualisations import plot_reliability_diagram

labels = np.array([0, 0, 0, 1, 0, 1, 1, 1])
scores = np.array([0.1, 0.2 ,0.3, 0.4, 0.6, 0.7, 0.8, 0.9])
bins = [0, 0.5, 1.0]
fig = plt.figure(figsize=(5, 4))
fig = plot_reliability_diagram(labels, np.vstack([1 - scores, scores]).T,
                               class_names=['not 1', 'rain'], bins=bins,
                               fig=fig, show_gaps=True,
                               show_bars=True)
```



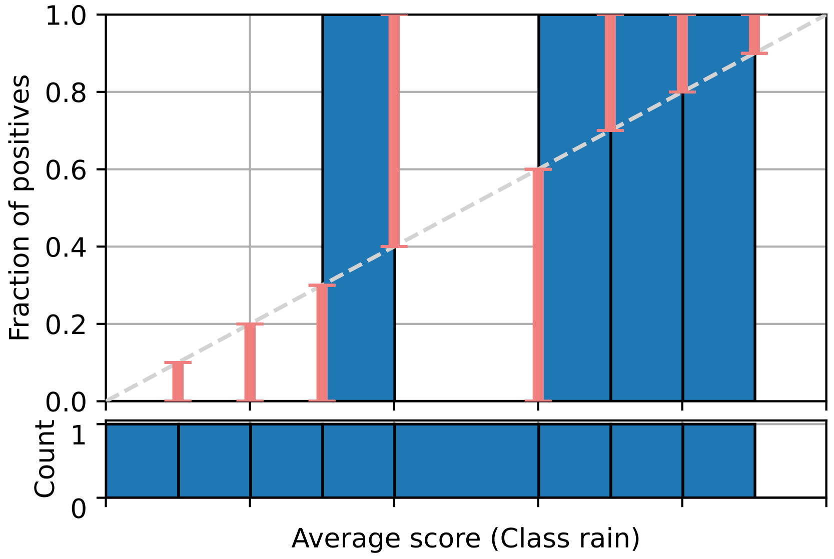
Or not at all?

	$\hat{p}$	y
0	0.1	0
1	0.2	0
2	0.3	0
3	0.4	1
4	0.6	0
5	0.7	1
6	0.8	1
7	0.9	1

```
import numpy as np
import matplotlib.pyplot as plt

from pycalib.visualisations import plot_reliability_diagram

labels = np.array([0, 0, 0, 1, 0, 1, 1, 1])
scores = np.array([0.1, 0.2 ,0.3, 0.4, 0.6, 0.7, 0.8, 0.9])
bins = [0, 0.101, 0.201, 0.301, 0.401, 0.601, 0.701, 0.801, 0.901, 1.0]
fig = plt.figure(figsize=(5, 4))
fig = plot_reliability_diagram(labels, np.vstack([1 - scores, scores]).T,
                               class_names=['not 1', 'rain'], bins=bins,
                               fig=fig, show_gaps=True,
                               show_bars=True)
```



Binning or pooling predictions is a fundamental notion

We need bins to **evaluate** the degree of calibration:

- In order to decide whether a weather forecaster is well-calibrated, we need to look at a good number of forecasts, say over one year.
- We also need to make sure that there are a reasonable number of forecasts for separate probability values, so we can obtain reliable empirical estimates.
    - Trade-off: large bins give better empirical estimates, small bins allows a more fine-grained assessment of calibration.}

But adjusting forecasts in groups also gives rise to practical calibration **methods**:

- empirical binning
- isotonic regression (aka ROC convex hull)

Questions and answers

Q&A 1

Question 1

A binary classifier for weather predictions produces a score of 0.1 for rain two times but it does not rain, two times 0.4 and it rains only once, five times 0.6 and it rains 80% of the times and one time 0.9 and it rains. Does the following reliability diagram show that information?

Answer: Yes

Correct. You can see that there is one bin per predicted score $0.1, 0.4, 0.6$ and 0.9. Each bin contains the number of scores indicated in the smaller histogram below with 2, 2, 5 and 1 samples respectively. Finally, the height of each bin corresponds to the fraction of rains indicated in the question 0%, 50%, 80% and 100%.

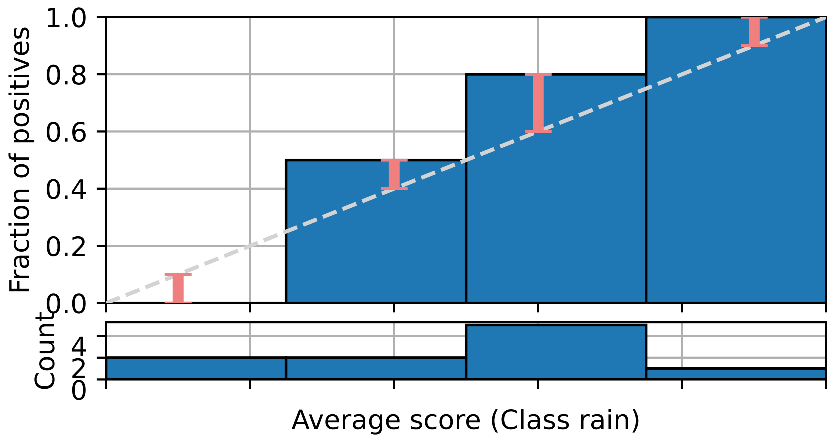
Answer: No

Incorrect. Try another answer.

```
import numpy as np
import matplotlib.pyplot as plt

from pycalib.visualisations import plot_reliability_diagram

labels = np.array([0,   0,  0,  1,  0,  1,  1,  1,  1,  1])
scores = np.array([.1, .1, .4, .4, .6, .6, .6, .6, .6, .9])
fig = plt.figure(figsize=(5, 3))
fig = plot_reliability_diagram(labels, np.vstack([1 - scores, scores]).T,
                               class_names=['not 1', 'rain'], bins=4,
                               fig=fig,
                               hist_per_class=False,
                               show_bars=True,
                               show_gaps=True)
```



Q&A 2

Question 2

A binary classifier for weather predictions produces a score of 0.1 for rain two times and it rains once, three times 0.4 and it rains two times, four times 0.6 and it rains once and one time 0.9 and it rains. Does the following reliability diagram show that information?

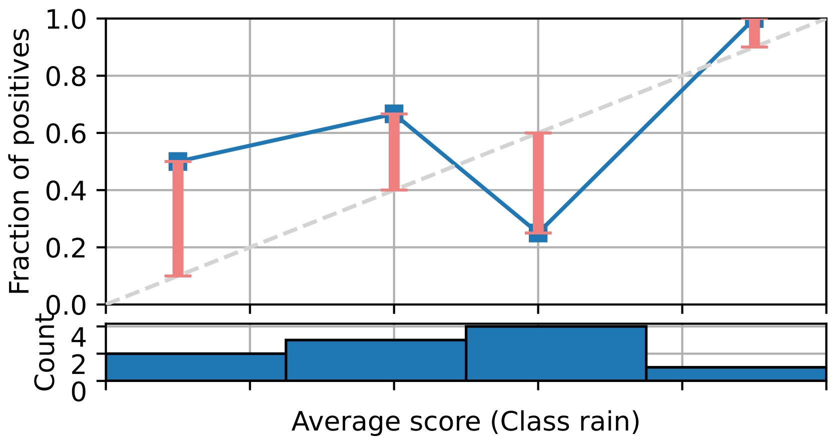
Answer: Yes

Correct. You can see that there is one bin per predicted score $0.1, 0.4, 0.6$ and 0.9. Each bin contains the number of scores indicated in the smaller histogram below with 2, 3, 4 and 1 samples respectively. Finally, the height of each bin corresponds to the fraction of rains indicated in the question 50%, 66.6%, 25% and 100%.

Answer: No

Incorrect. Try another answer.

```
labels = np.array([0,   1,  0,  1,  1,  0,  0,  0,  1,  1])
scores = np.array([.1, .1, .4, .4, .4, .6, .6, .6, .6, .9])
fig = plt.figure(figsize=(5, 3))
fig = plot_reliability_diagram(labels, np.vstack([1 - scores, scores]).T,
                               class_names=['not 1', 'rain'], bins=4,
                               fig=fig,
                               hist_per_class=False,
                               show_bars=False,
                              show_gaps=True)
```



Q&A 3

Question 3

Do we need multiple instances in each bin in order to visualise a reliability diagram?

Answer: Yes

Incorrect. Try another answer.

Answer: No

Correct. It is not necessary to have multiple instances in each bin for visualisation purposes. However, the lack of information does not allow us to know if the model is calibrated for those scores.

Q&A 4

Question 4

The following figure shows the reliability diagram of a binary classifier on enough samples to be statistically significant. Is the model calibrated, producing under-estimates or over-estimates?

Answer: Over-estimates

Incorrect. Try another answer.

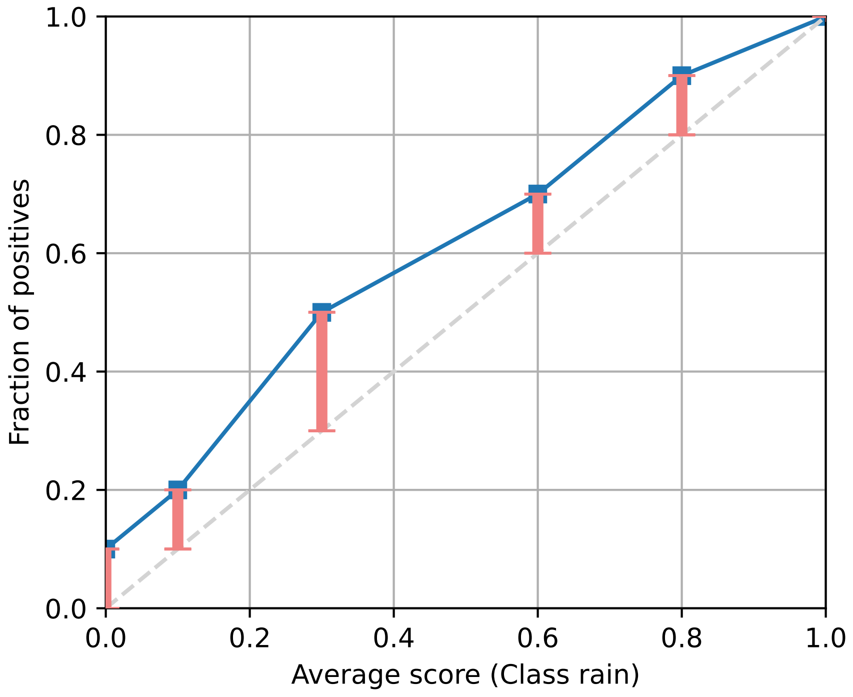
Answer: Under-estimates

Correct. For each predicted score the actual fraction of positives is higher.

```
import numpy as np

from pycalib.visualisations import plot_reliability_diagram_precomputed

scores = np.array([0, .1, .3, .6, .8, 1]).reshape(-1, 1)
empirical = np.array([.1, .2, .5, .7, .9, 1]).reshape(-1, 1)
fig = plt.figure(figsize=(5, 4))
fig = plot_reliability_diagram_precomputed(avg_true=empirical, avg_pred=scores,
                                           class_names=['rain'],
                                           fig=fig, show_gaps=True)
```



Q&A 5

Question 5

The following figure shows the reliability diagram of a binary classifier on enough samples to be statistically significant. Is the model calibrated, producing under-estimates or over-estimates?

Answer: Over-estimates

Correct. For each predicted score the actual fraction of positives is lower.

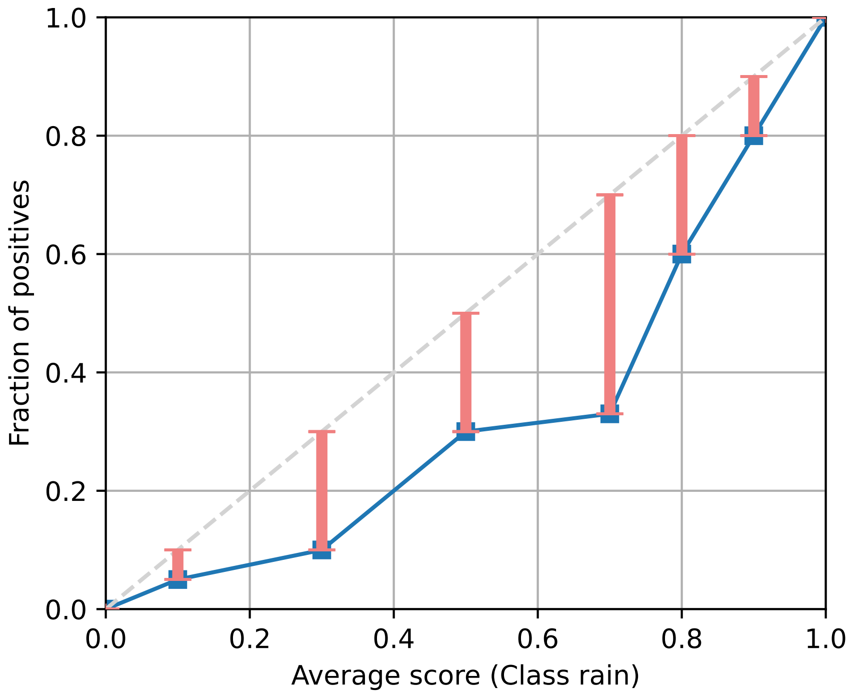
Answer: Under-estimates

Incorrect. Try another answer.

```
import numpy as np

from pycalib.visualisations import plot_reliability_diagram_precomputed

scores = np.array([0, .1, .3, .5, .7, .8, .9, 1]).reshape(-1, 1)
empirical = np.array([0, .05, .1, .3, .33, .6, .8, 1]).reshape(-1, 1)
fig = plt.figure(figsize=(5, 4))
fig = plot_reliability_diagram_precomputed(avg_true=empirical, avg_pred=scores,
                                           class_names=['rain'],
                                           fig=fig, show_gaps=True)
```



Why are we interested in calibration?

Why are we interested in calibration?

To calibrate means **to employ a known scale with known properties**.

- E.g., additive scale with a well-defined zero, so that ratios are meaningful.

For classifiers we want to use the probability scale, so that we can

- justifiably use default decision rules (e.g., maximum posterior probability);
- adjust these decision rules in a straightforward way to account for different class priors or misclassification costs;
- combine probability estimates in a well-founded way.

Q: Is the probability scale additive?

In some situations we may want to sum probabilities, for example if we have a set of mutually exclusive events, the probability of at least one of them happening can be computed by their sum. In other situations the product of probabilities is used, e.g. the probability of two independent events happening at the same time.

Q: How would you combine probability estimates from several well-calibrated models?

Check some online information e.g. When pooling forecasts, use the geometric mean of odds
And the following code shows some examples.

```
import numpy as np
from tabulate import tabulate
from IPython.display import Markdown

def mean(p):
    '''Arithmetic mean'''
    p = np.array(p)
    return np.sum(p)/(len(p))

def gmean(p):
    '''Geometric mean'''
    p = np.array(p)
    o = np.power(np.prod(p/(1-p)), 1/len(p))
    return o/(1+o)

def hmean(p):
    '''Harmonic mean'''
    p = np.array(p)
    return len(p)/np.sum(1/p)

example_list = [[.1, .1], [.5, .5], [.1, .9],
                [.1, .1, .9], [.1, .1, .99], [.1, .1, .999]]

functions = {'Arithmetic mean': mean,
             'Geometric mean': gmean,
             'Harmonic mean': hmean}

table = []

for example in example_list:
    table.append([np.array2string(np.array(example))])
    table[-1].extend([f'{f(example):.2f}' for f in functions.values()])


headers = ['Probabilities']
headers.extend(list(functions.keys()))
```

```
Markdown(tabulate(table, headers=headers))
```

Table 6: Example of probability means

Probabilities	Arithmetic mean	Geometric mean	Harmonic mean
[0.1 0.1]	0.1	0.1	0.1
[0.5 0.5]	0.5	0.5	0.5
[0.1 0.9]	0.5	0.5	0.18
[0.1 0.1 0.9]	0.37	0.32	0.14
[0.1 0.1 0.99]	0.4	0.52	0.14
[0.1 0.1 0.999]	0.4	0.7	0.14

Optimal decisions I

Denote the cost of predicting class j for an instance of true class i as $C(\hat{Y} = j|Y = i)$. The expected cost of predicting class j for instance x is

$$C(\hat{Y} = j|X = x) = \sum_i P(Y = i|X = x)C(\hat{Y} = j|Y = i)$$

where $P(Y = i|X = x)$ is the probability of instance x having true class i (as would be given by the Bayes-optimal classifier).

The optimal decision is then to predict the class with lowest expected cost:

$$\hat{Y}^* = \arg\min_j C(\hat{Y} = j|X = x) = \arg\min_j \sum_i P(Y = i|X = x)C(\hat{Y} = j|Y = i)$$

Optimal decisions II

In binary classification we have:

$$\begin{aligned} C(\hat{Y} = +|X = x) &= P(+|x)C(+|+) + (1 - P(+|x))C(+|-) \\ C(\hat{Y} = -|X = x) &= P(+|x)C(-|+) + (1 - P(+|x))C(-|-) \end{aligned}$$

On the optimal decision boundary these two expected costs are equal, which gives

$$P(+|x) = \frac{C(+|-) - C(-|-)}{C(+|-) - C(-|-) + C(-|+) - C(+|+)} \triangleq c$$

This gives the optimal threshold on the hypothetical Bayes-optimal probabilities.

It is also the best thing to do in practice – as long as the probabilities are well-calibrated!

Optimal decisions III

Without loss of generality we can set the cost of true positives and true negatives to zero; $c = \frac{c_{\text{FP}}}{c_{\text{FP}}+c_{\text{FN}}}$ is then the cost of a false positive in proportion to the combined cost of one false positive and one false negative.

- E.g., if false positives are 4 times as costly as false negatives then we set the decision threshold to $4/(4+1) = 0.8$ in order to only make positive predictions if we're pretty certain.

Similar reasoning applies to changes in class priors:

- if we trained on balanced classes but want to deploy with 4 times as many positives compared to negatives, we lower the decision threshold to 0.2;
- more generally, if we trained for class ratio r and deploy for class ratio r' we set the decision threshold to $r/(r+r')$.

Cost and class prior changes can be combined in the obvious way.

Questions and answers

Q&A 1

Question 1

Is it possible to compute optimal risks given a cost matrix and a probabilistic classifier that is not calibrated?

Answer: Yes

Incorrect. Try another answer.

Answer: No

Correct.

Q&A 2

Question

Given a calibrated probabilistic classifier, is it optimal to select the class with the highest predicted probability?

Answer: Yes

Incorrect. Try another answer.

Answer: No

Correct.

Question

If we have the following cost matrix, and a model outputs 0.6 probability for class 1. What would be the expected cost of predicting class 2?

```python
import numpy as np
from tabulate import tabulate
from IPython.display import Markdown

cost_matrix = [[-1,  4],
               [ 2, -2]]

table = [['True Class 1'],
         ['True Class 2']]

for i, c in enumerate(cost_matrix):
    table[i].extend(c)

headers = ['Predicted Class 1', 'Predicted Class 2']

Markdown(tabulate(table, headers=headers))
```

Table 7: Example of a cost matrix

	Predicted Class 1	Predicted Class 2
True Class 1	-1	4
True Class 2	2	-2

Answer: 0.4

Incorrect. Try another answer.

Answer: -0.4

Incorrect. Try another answer.

Answer: 1.6

Correct. $4 * 0.6 - 2 * 0.4$

Question

What would be the expected cost of predicting class 1?

Answer: 0.4

Incorrect. Try another answer.

Answer: -0.4

Correct. $-1 * 0.6 + 2 * 0.4$

Answer: 1.6

Incorrect. Try another answer.

Common sources of miscalibration

Common sources of miscalibration

Underconfidence: a classifier thinks it's **worse** at separating classes than it actually is.

- Hence we need to *pull predicted probabilities away from the centre.*

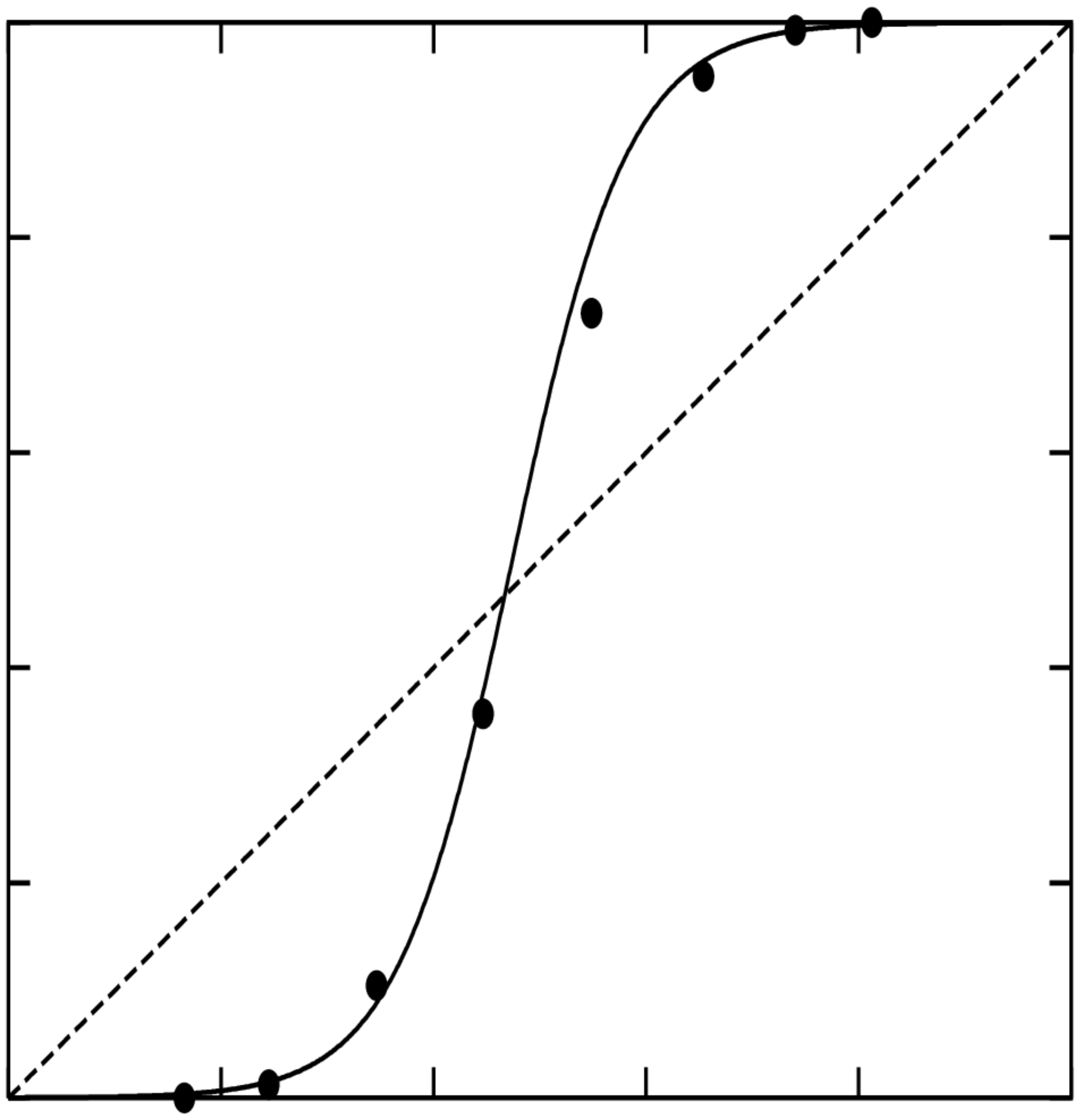
Overconfidence: a classifier thinks it's **better** at separating classes than it actually is.

- Hence we need to *push predicted probabilities toward the centre.*

A classifier can be overconfident for one class and underconfident for the other, in which case all predicted probabilities need to be increased or decreased.

Underconfidence example

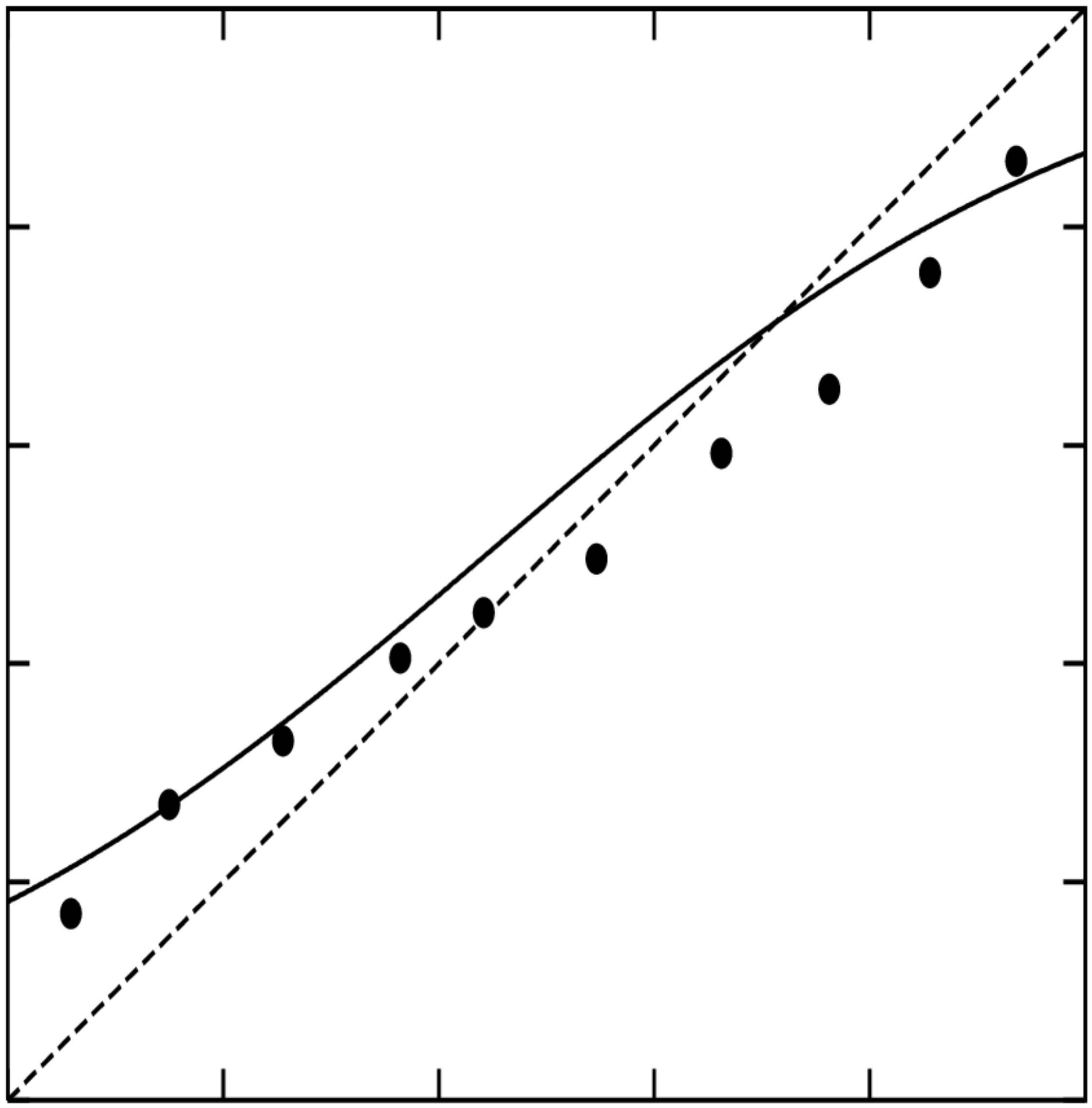
- Underconfidence typically gives *sigmoidal* distortions.
- To calibrate these means to *pull predicted probabilities away from the centre.*



Source: (Niculescu-Mizil and Caruana 2005)

Overconfidence example

- Overconfidence is very common, and usually a consequence of over-counting evidence.
- Here, distortions are *inverse-sigmoidal*
- Calibrating these means to *push predicted probabilities toward the centre*.



Source: (Niculescu-Mizil and Caruana 2005)

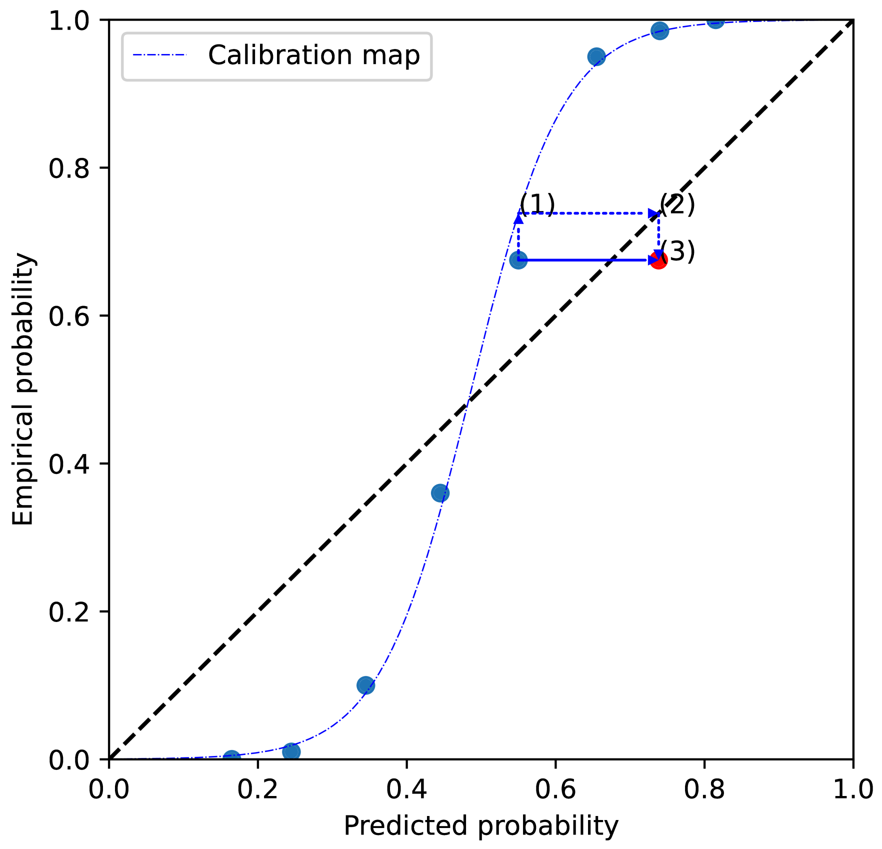
Why fitting the distortions helps with calibration

In clockwise direction, the dotted arrows show:

1. using a point's uncalibrated score on the x-axis as input to the calibration map,
2. mapping the resulting output back to the diagonal, and

3. combine with the empirical probability of the point we started from.

The closer the original point is to the fitted calibration map, the closer the calibrated point (in red) will be to the diagonal.



Questions and answers

Q&A 1

Question

The following figures show the reliability diagram of several binary classifiers. Assuming that there are enough samples on each bin, indicate if the model seems calibrated, over-confident or under-confident.

Answer: Calibrated.

Incorrect. Try another answer.

Answer: Over-confident.

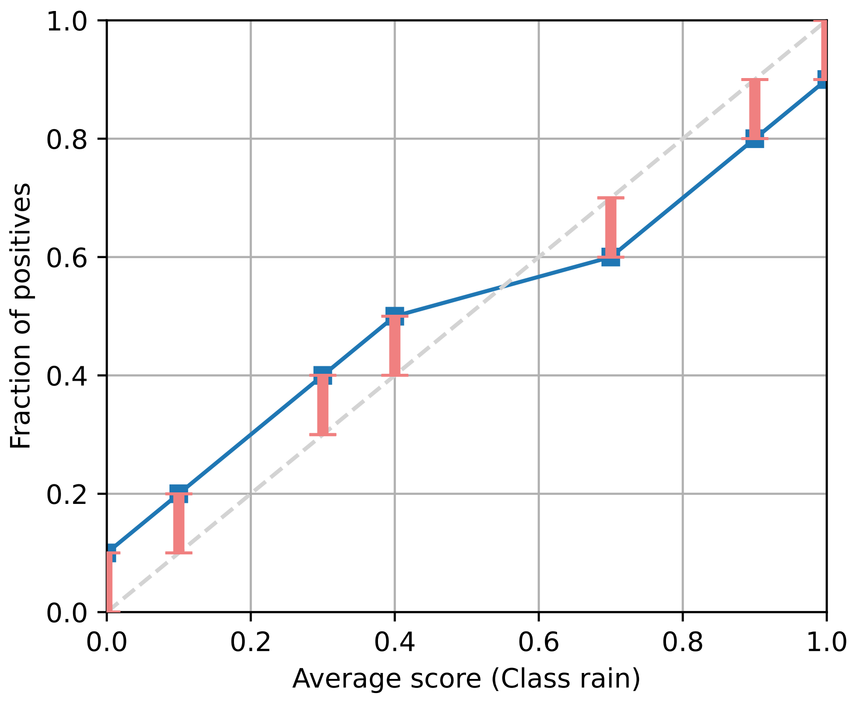
Correct.

Answer: Under-confident.

Incorrect. Try another answer.

```python
import numpy as np
from pycalib.visualisations import plot_reliability_diagram_precomputed

scores =    np.array([0., .1, .3, .4, .7, .9, 1.]).reshape(-1, 1)
empirical = np.array([.1, .2, .4, .5, .6, .8, .9]).reshape(-1, 1)
fig = plt.figure(figsize=(5, 4))
fig = plot_reliability_diagram_precomputed(avg_true=empirical, avg_pred=scores,
                                           class_names=['rain'],
                                           fig=fig, show_gaps=True)
```



Q&A 2

Answer: Calibrated.

Correct.

Answer: Over-confident.

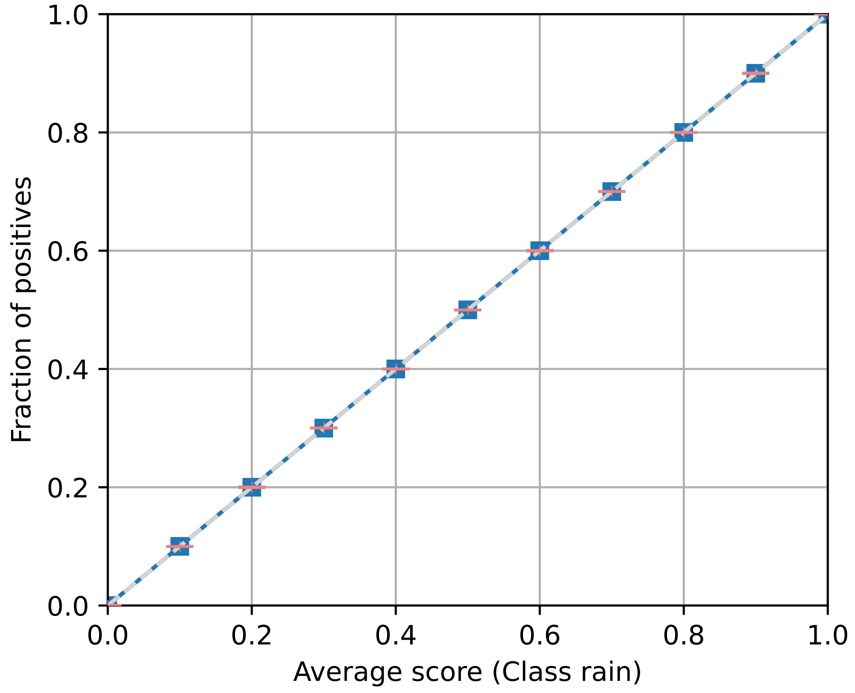
Incorrect. Try another answer.

Answer: Under-confident.

Incorrect. Try another answer.

```
import numpy as np
from pycalib.visualisations import plot_reliability_diagram_precomputed

scores = np.array([0, .1, .2, .3, .4, .5, .6, .7, .8, .9, 1]).reshape(-1, 1)
empirical = scores
fig = plt.figure(figsize=(5, 4))
fig = plot_reliability_diagram_precomputed(avg_true=empirical, avg_pred=scores,
                               class_names=['rain'],
                               fig=fig, show_gaps=True)
```



Q&A 3

Answer: Calibrated.

Incorrect. Try another answer.

Answer: Over-confident.

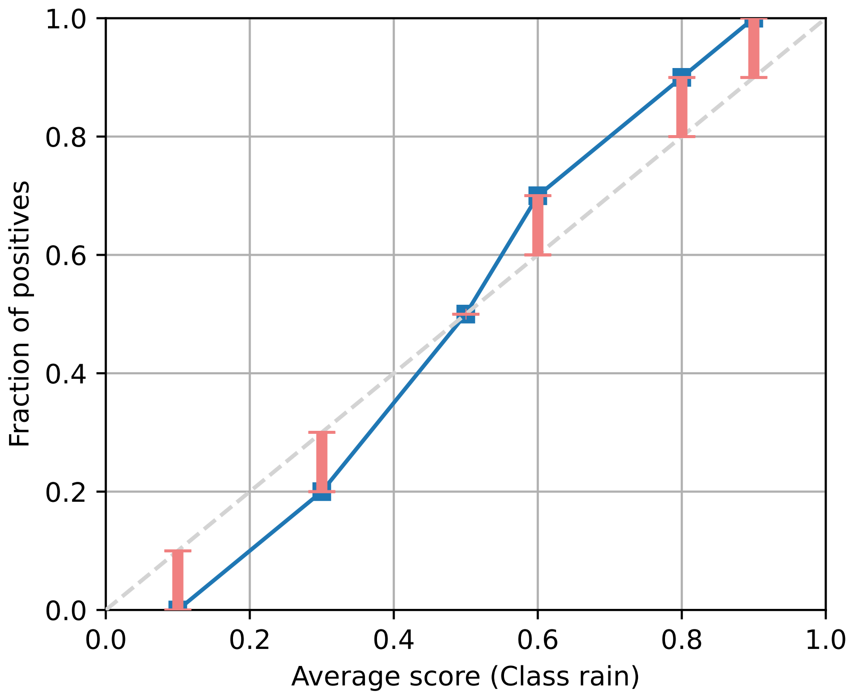
Incorrect. Try another answer.

Answer: Under-confident.

Correct.

```
import numpy as np
from pycalib.visualisations import plot_reliability_diagram_precomputed
```

```
scores =    np.array([.1, .3, .5, .6, .8, .9]).reshape(-1, 1)
empirical = np.array([0., .2, .5, .7, .9, 1.]).reshape(-1, 1)
fig = plt.figure(figsize=(5, 4))
fig = plot_reliability_diagram_precomputed(avg_true=empirical, avg_pred=scores,
                                            class_names=['rain'],
                                            fig=fig, show_gaps=True)
```



Q&A 4

Question

Can a binary classifier show a calibrated reliability diagram with a number of equally distributed bins, and a non-calibrated one with a higher number of equally distributed bins?

Answer: Yes.

Correct.

Answer: No.

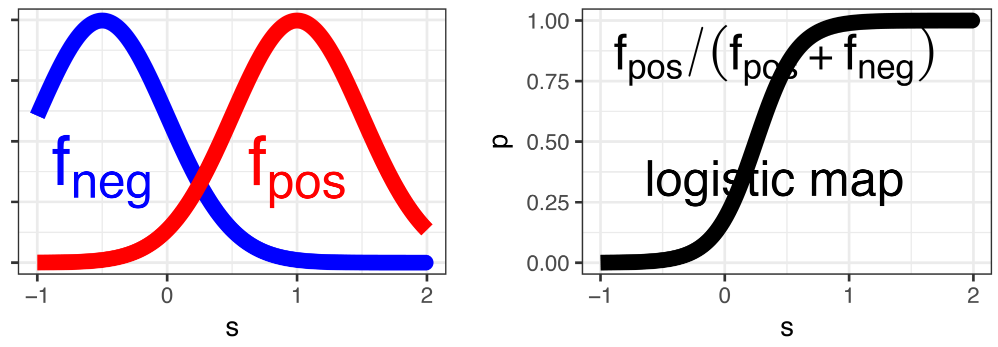
Incorrect. Try another answer.

A first look at some calibration techniques

A first look at some calibration techniques

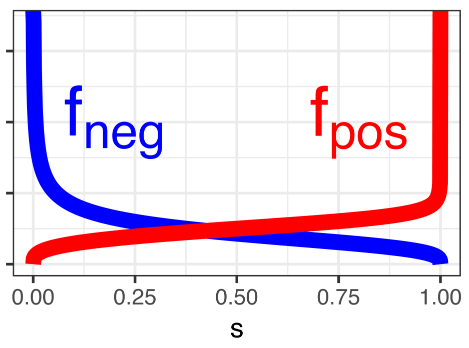
- **Parametric** calibration involves modelling the score distributions within each class. \
  - **Platt scaling** = Logistic calibration can be derived by assuming that the scores within both classes are normally distributed with the same variance (Platt 2000).
  - **Beta calibration** employs Beta distributions instead, to deal with scores already on a $[0, 1]$ scale (Kull, Silva Filho, and Flach 2017).
  - **Dirichlet calibration** for more than two classes (Kull et al. 2019).
- **Non-parametric** calibration often ignores scores and employs ranks instead. \
  - E.g., **isotonic regression** = pool adjacent violators = ROC convex hull (Zadrozny and Elkan 2001) (Fawcett and Niculescu-Mizil 2007).

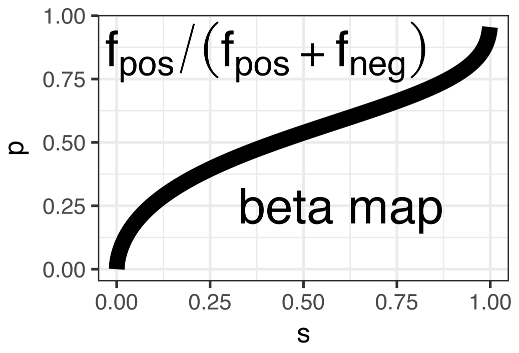
Platt scaling



$$p(s; w, m) = \frac{1}{1 + \exp(-w(s - m))}$$
$$w = (\mu_{pos} - \mu_{neg})/\sigma^2, m = (\mu_{pos} + \mu_{neg})/2$$

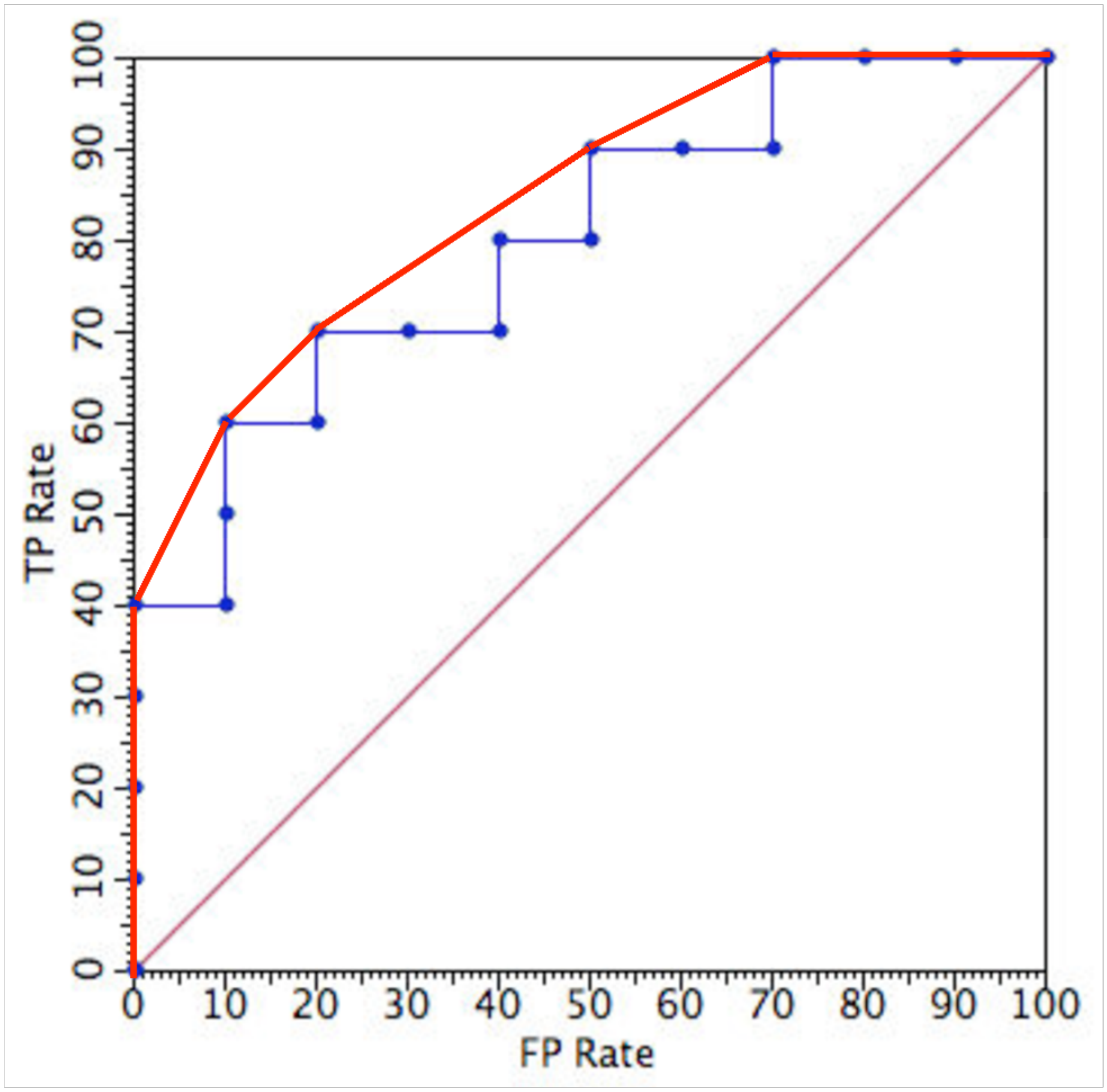
Beta calibration

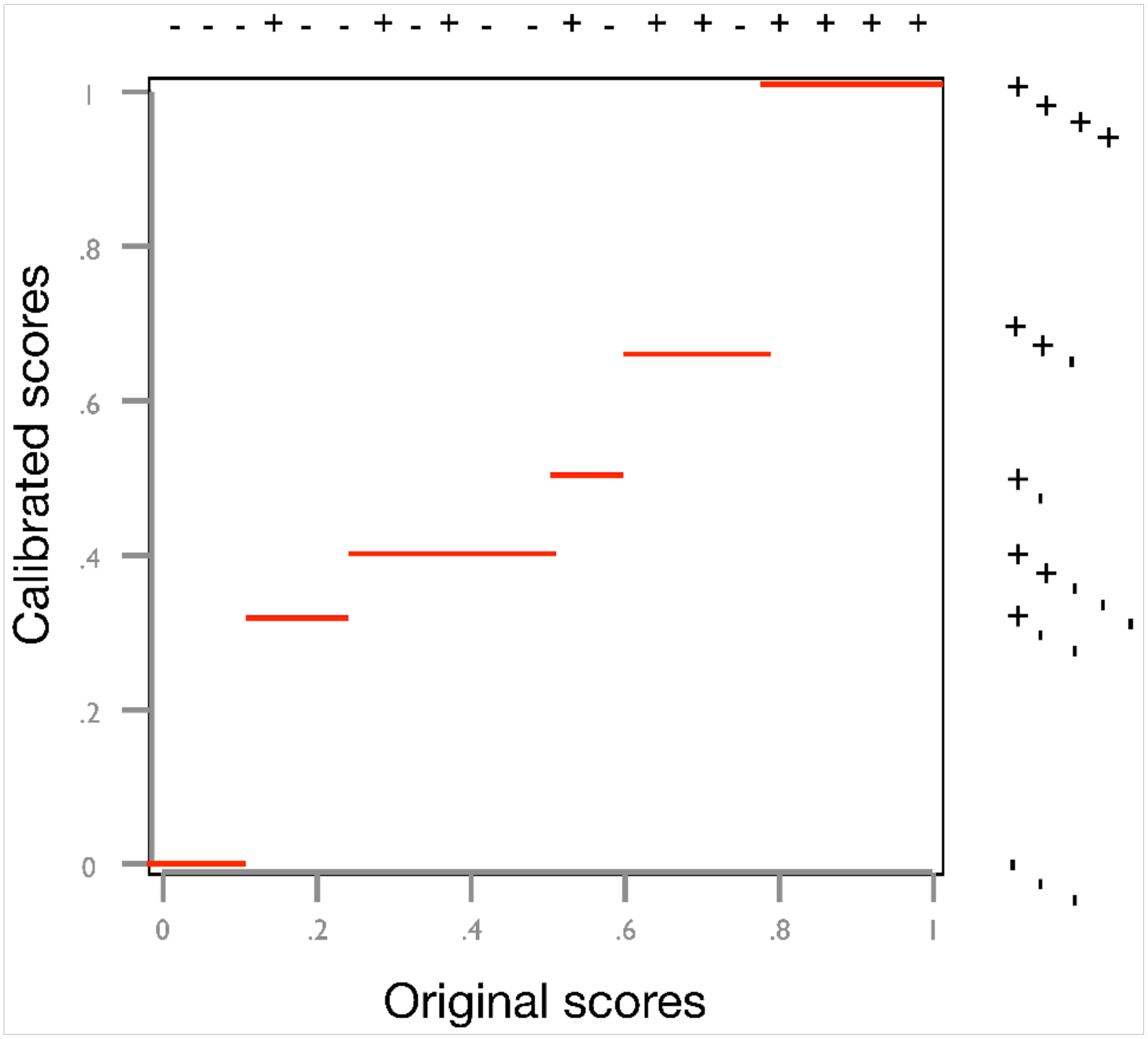




$$p(s;a,b,c) = \frac{1}{1 + \exp(-a \ln s - b \ln(1-s) - c)}$$
$$a = \alpha_{pos} - \alpha_{neg}, b = \beta_{neg} - \beta_{pos}$$

Isotonic regression





Source: Flach (2016)

Questions and answers

Question

Can a binary classifier show a calibrated reliability diagram with a number of equally distributed bins, and a non-calibrated one with a higher number of equally distributed bins?

Answer: What is a calibration map?

It is a mapping between the scores to be calibrated and the objective probabilities.

Platt scaling:

Can Platt scaling calibrate probabilistic models that are overconfident?

Answer: Yes.

Incorrect. Try another answer.

Answer: No.

Correct. Platt scaling for probability scores can only generate S shaped calibration maps, which can only calibrate under-confident scores.

Isotonic regression:

Is isotonic regression a parametric method?

Answer: Yes.

Incorrect. Try another answer.

Answer: No.

Correct.

Platt scaling:

Can Platt scaling learn an identity function if the model is already calibrated?

Answer: Yes.

Incorrect. Try another answer.

Answer: No.

Correct. Platt scaling can only learn S shaped functions, and the identity function requires a straight line.

Calibrating multi-class classifiers

What's so special about multi-class calibration?

Similar to classification, some methods are inherently multi-class but most are not.

This leads to (at least) three different ways of **defining** what it means to be fully multiclass-calibrated. - Many recent papers use the (weak) notion of confidence calibration.

Evaluating multi-class calibration is in its full generality still an open problem.

Definitions of calibration for more than two classes

The following definitions of calibration are equivalent for binary classification but increasingly stronger for more than two classes:

- **Confidence calibration:** only consider the highest predicted probability.
- **Class-wise calibration:** only consider marginal probabilities.
- **Multi-class calibration:** consider the entire vector of predicted probabilities.

Confidence calibration

This was proposed by Guo et al. (2017), requiring that among all instances where the probability of **the most likely class** is predicted to be c, the expected accuracy is c. (We call this 'confidence calibration' to distinguish it from the stronger notions of calibration.)

Formally, a probabilistic classifier $\hat{\mathbf{p}} : \mathcal{X} \to \Delta_k$ is **confidence-calibrated**, if for any confidence level $c \in [0, 1]$, the actual proportion of the predicted class, among all possible instances $\mathbf{x}$ being predicted this class with confidence c, is equal to c:

$$P(Y = i \mid \hat{p}_i(\mathbf{x}) = c) = c \qquad \text{where} \quad i = \arg\max_j \hat{p}_j(\mathbf{x}).$$

Class-wise calibration

Originally proposed by Zadrozny and Elkan (2002), this requires that all **one-vs-rest** probability estimators obtained from the original multiclass model are calibrated.

Formally, a probabilistic classifier $\hat{\mathbf{p}} : \mathcal{X} \to \Delta_k$ is **classwise-calibrated**, if for any class i and any predicted probability q_i for this class, the actual proportion of class i, among all possible instances $\mathbf{x}$ getting the same prediction $\hat{p}_i(\mathbf{x}) = q_i$, is equal to q_i:

$$P(Y = i \mid \hat{p}_i(\mathbf{x}) = q_i) = q_i \qquad \text{for } \; i = 1, \dots, k.$$

Multi-class calibration

This is the **strongest form of calibration** for multiple classes, subsuming the previous two definitions.

A probabilistic classifier $\hat{\mathbf{p}} : \mathcal{X} \to \Delta_k$ is **multiclass-calibrated** if for any prediction vector $\mathbf{q} = (q_1, \dots, q_k) \in \Delta_k$, the proportions of classes among all possible instances $\mathbf{x}$ getting the same prediction $\hat{\mathbf{p}}(\mathbf{x}) = \mathbf{q}$ are equal to the prediction vector $\mathbf{q}$:

$$P(Y = i \mid \hat{\mathbf{p}}(\mathbf{x}) = \mathbf{q}) = q_i \qquad \text{for } \; i = 1, \dots, k.$$

Reminder: binning needed

For practical purposes, the conditions in these definitions need to be relaxed. This is where **binning** comes in.

Once we have the bins, we can draw a **reliability diagram** as in the two-class case. For class-wise calibration, we can show per-class reliability diagrams or a single averaged one.

The degree of calibration is assessed using the **gaps** in the reliability diagram. All of this will be elaborated in the next part of the tutorial.

Important points to remember

- **Only well-calibrated probability estimates are worthy to be called probabilities:** otherwise they are just scores that happen to be in the $[0, 1]$ range.
- **Binning will be required in some form:** instance-based probability evaluation metrics such as Brier score or log-loss always measure calibration **plus something else**.
- **In multi-class settings, think carefully about which form of calibration you need:** e.g., confidence-calibration is too weak in a cost-sensitive setting.

Questions and answers

Model scores:

Can we interpret the output of any model that produces values between zero and one as probabilities?

Answer: Yes.

Incorrect. Try another answer.

Answer: No.

Correct. Some models have been trained to generate values in any arbitrary range, but this does not mean that the model is predicting actual probabilities.

Question

Is there only one way to measure calibration for multiclass probability scores.

Answer: Yes.

Incorrect. Try another answer.

Answer: No.

Correct. There are multiple measures of calibration.

Question

Can we perform optimal decisions in a multiclass setting by knowing the highest probability among the classes and the misclassification costs?

Answer: Yes.

Incorrect. Try another answer.

Answer: No.

Correct. We need the probabilities of every class in order to make an optimal decision.

Question

If the data distribution, operating conditions and the missclassification costs do not change from training to test set, and a model makes optimal predictions in the training set. Do we need the exact probabilities in the test set to make optimal decisions?

Answer: Yes.

Incorrect. Try another answer.

Answer: No.

Correct.

References

Brier, Glenn W. 1950. "Verification of forecasts expressed in terms of probabilities." *Monthly Weather Review* 78 (1): 1–3.

Fawcett, Tom, and Alexandru Niculescu-Mizil. 2007. "PAV and the ROC convex hull." *Machine Learning* 68 (1): 97–106.

Flach, Peter A. 2016. "ROC Analysis." In *Encyclopedia of Machine Learning and Data Mining*. Springer.

Guo, Chuan, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. 2017. "On Calibration of Modern Neural Networks." In *34th International Conference on Machine Learning*, 1321–30.

Kull, Meelis, Miquel Perello-Nieto, Markus Kängsepp, Telmo Silva Filho, Hao Song, and Peter Flach. 2019. "Beyond temperature scaling: Obtaining well-calibrated multiclass probabilities with Dirichlet calibration." In *Advances in Neural Information Processing Systems (NIPS'19)*, 12316–26.

Kull, Meelis, Telmo M. Silva Filho, and Peter Flach. 2017. "Beyond Sigmoids: How to obtain well-calibrated probabilities from binary classifiers with beta calibration." *Electronic Journal of Statistics* 11 (2): 5052–80.

Niculescu-Mizil, Alexandru, and Rich Caruana. 2005. "Predicting good probabilities with supervised learning." In *22nd International Conference on Machine Learning (ICML'05)*, 625–32. ACM Press.

Platt, JC. 2000. "Probabilities for SV Machines." In *Advances in Large-Margin Classifiers*, edited by Alexander J. Smola, Peter Bartlett, Bernhard Schölkopf, and Dale Schuurmans, 61—–74. MIT Press.

Zadrozny, Bianca, and Charles Elkan. 2001. "Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers." In *18th International Conference on Machine Learning (ICML'01)*, 609—–616.

———. 2002. "Transforming Classifier Scores into Accurate Multiclass Probability Estimates."

In *8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '02*, 694——699. ACM Press.