Introduction to Optimal Decision Making

Miquel Perello Nieto

Contents

1 Introduction

This is an introductory course on the topic of `Optimal Decision Making` for multiclass classification. In Section 2, we give some motivation insights about the importance of estimating missclassification costs when making automatic decisions, then we define a common setting to calculate expected costs and how to make an optimal decision based on those expected costs in the binary (Section 3.1) and multiclass (Section 3.6) case. We provide interactive examples that demonstrate how the optimal decision change under different operational conditions. Finally in Section 3.8, we demonstrate examples in which abstaining on making a prediction and delegating the decision may be optimal solution.

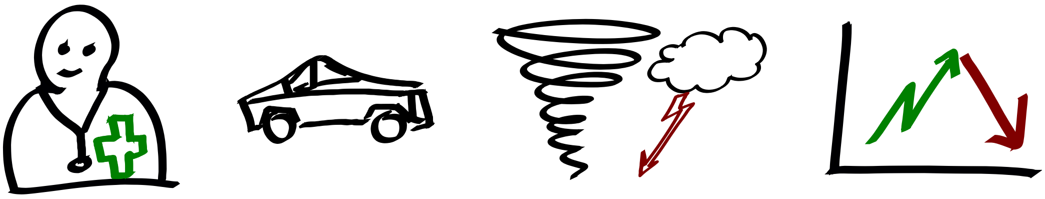
2 Motivation

2.1 Optimal Decision Making. Why and how?

One of the most common objectives of multiclass classification is to train a machine learning model that is able to predict the most probable class given a new instance. However, in certain applications in which the missclassification costs are of main importance, the most probable class may not be the one that provides the highest expected utilities. The objective of optimal decision making is to predict the class that maximises the expected utilities, and minimises the expected misclassification costs. Some typical scenarios in which this is important are medical diagnosis, self-driving cars, extreme weather prediction and finances.

2.1.1 Key points

- The objective is to classify a new instance into one of the possible classes in an optimal manner.
- This may be important in critical applications: e.g. medical diagnosis [2, 16], self-driving cars [14, 11], extreme weather prediction, finances [12].



- It is necessary to know what are the consequences of making each prediction (costs or gains).
- One way to make optimal decisions is with cost-sensitive classification.
- Can we make optimal decisions with any type of classifier?

2.2 Optimal decisions with different types of model

- **Class estimation**: Outputs a class prediction.
- **Class estimation with option of abstaining**: Outputs a class prediction or abstains [5, 10]

- **Rankings estimation**: Outputs a ranked list of possible classes [3].
- **Score surrogates**: Outputs a continuous score which is commonly a surrogate for classification (e.g. Support Vector Machines).
- **Probability estimation**: Outputs class posterior probability estimates (e.g. Logistic Regression, naive Bayes, Artificial Neural Networks), or provides class counts which can be interpreted as proportions (e.g. decision trees, random forests, k-nearest neightbour) [17].
- **Other types of outputs**: Some examples are possibility theory [7], credal sets [9], conformal predictions [15], multi-label [1].

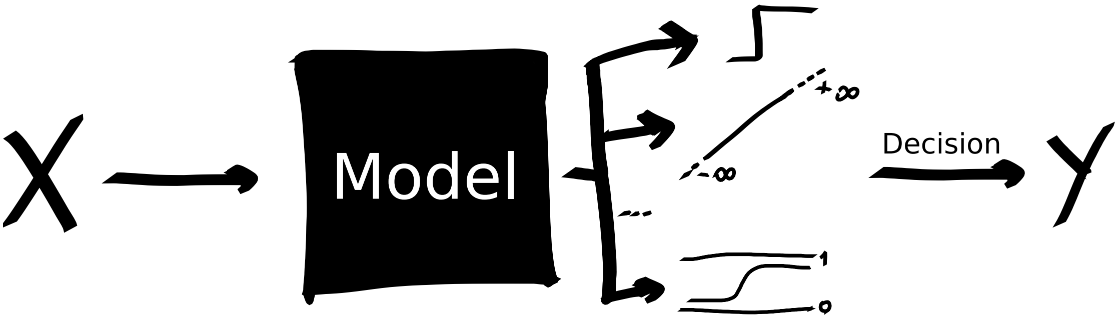


Figure 2.1: Classifier as a black box

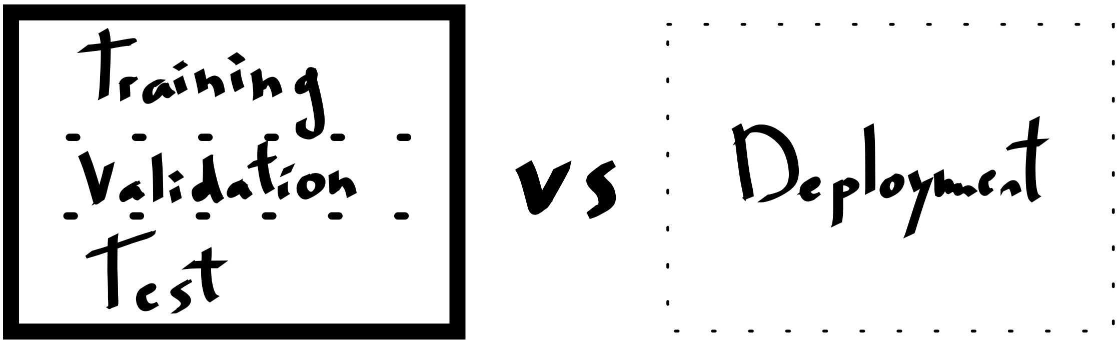


Figure 2.2: Training vs Deployment

3 Cost-sensitive classification

- Cost-sensitive classification [8] provides a framework to make optimal decisions (with certain assumptions).
- We require the true posterior probabilities of each outcome in order to make optimal decisions, but we can use estimates.
- Assumes the costs are not instance dependent (only depend on the predicted and true class).
- Class priors and costs can be changed during deployment (if known or estimated).

3.1 Cost matrices: Binary example

The following is a typical example of a cost matrix c for a binary problem.

	Predicted C_1	Predicted C_2
True C_1	0	1
True C_2	1	0

We will refer to $c_{i|j}$ the cost of predicting class C_i given that the true class is C_j.

Given the posterior probabilities $P(C_j|\mathbf{x})$ where $j \in \{1, K\}$ and the cost matrix c we can calculate the expected cost of predicting class C_i

$$\mathbb{E}_{j \sim P(\cdot|\mathbf{x})}(c_{i|j}) = \sum_{j=1}^{K} P(C_j|\mathbf{x}) c_{i|j}. \tag{3.1}$$

For example, lets assume that the posterior probability vector for a given instance is $[0.4, 0.6]$, the expected costs will be

- Predicting **Class 1** will have an expected cost of $0.4 \times 0 + 0.6 \times 1 = 0.6$
- Predicting **Class 2** will have an expected cost of $0.4 \times 1 + 0.6 \times 0 = \mathbf{0.4}$.

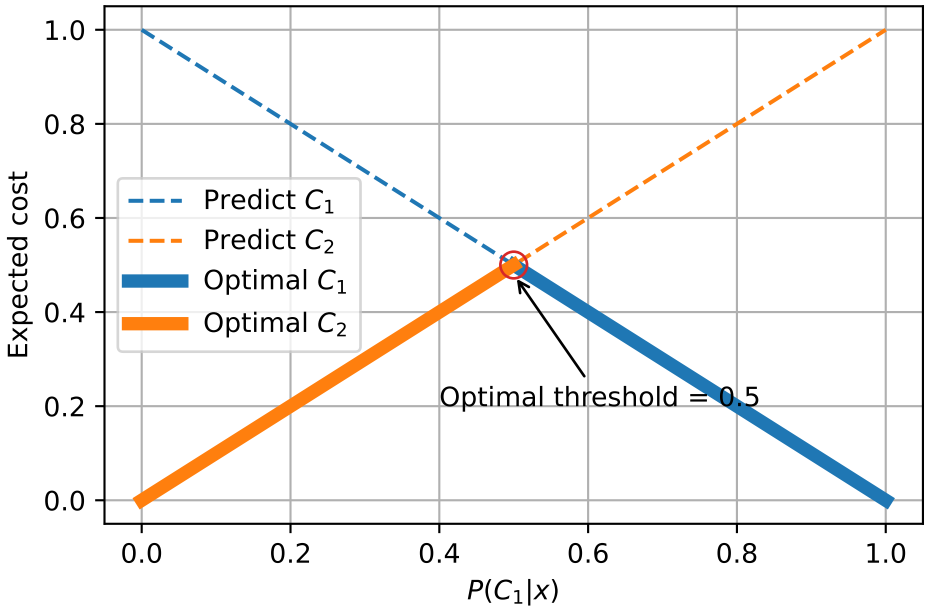
3.1.1 Expected costs figure

We can visualise the cost lines for each prediction with a line for each predicted class C_i and its missclassification costs and correct predictions [6]. For example, the following cost matrix

	Predicted C_1	Predicted C_2
True C_1	0	1
True C_2	1	0

will result in the following cost lines

```
import matplotlib.pyplot as plt

C = [[0, 1], [1, 0]]
threshold = (C[0][1] - C[1][1])/(C[0][1] - C[1][1] + C[1][0] - C[0][0])
cost_t = threshold*C[0][0] + (1-threshold)*C[0][1]
plt.grid(True)
plt.plot([0, 1], [C[0][1], C[0][0]], '--', label="Predict $C_1$")
plt.plot([0, 1], [C[1][1], C[1][0]], '--', label="Predict $C_2$")
plt.plot([threshold, 1], [cost_t, C[0][0]], lw=5, color='tab:blue', label="Optimal
plt.plot([0, threshold], [C[1][1], cost_t], lw=5, color='tab:orange', label="Optima
plt.xlabel('$P(C_1|x)$')
plt.ylabel('Expected cost')
plt.legend()
plt.annotate("Optimal threshold = 0.5", (0.5, 0.48), xytext=(0.4, 0.2),
             arrowprops=dict(arrowstyle='->', facecolor='black'))
plt.scatter(0.5, 0.5, s=100, facecolors='none', edgecolors='tab:red', zorder=10)
plt.show()
```



where we have highlighted the minimum cost among the possible predictions. In this particular case the optimal prediction changes when the probability of the true class is higher or lower than 0.5, with the same expected cost for both classes at 0.5.

3.2 Cost Matrix “reasonableness” condition

In general, it is reasonable to expect cost matrices where:

1. For a given class j the correct prediction has always a lower cost than an incorrect prediction $c_{j|j} < c_{i|j}$ with $i \neq j$.
2. **Class domination**: One class does not consistently have lower costs than other classes $c_{i|j} \leq c_{k|j}$ for all j.

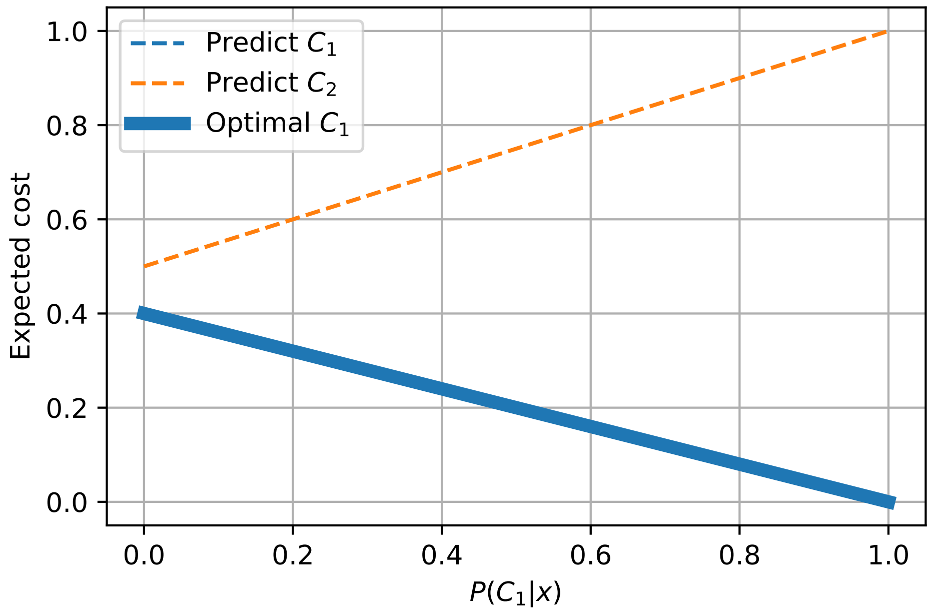
We will make these reasonable assumptions in this introductory module.

3.2.1 Class Domination

The following is an example of class domination in which predicting class C_1 will always have a lower expected cost.

	Predicted C_1	Predicted C_2
True C_1	0	1
True C_2	0.4	0.5

```python
import matplotlib.pyplot as plt

plt.grid(True)
plt.plot([0, 1], [0.4, 0], '--', color='tab:blue', label="Predict $C_1$")
plt.plot([0, 1], [0.5, 1], '--', color='tab:orange', label="Predict $C_2$")
plt.plot([0, 1], [0.4, 0], lw=5, color='tab:blue', label="Optimal $C_1$")
plt.xlabel('$P(C_1|x)$')
plt.ylabel('Expected cost')
plt.legend()
plt.show()
```



3.3 Optimal threshold for the binary case

If we know the true posterior probabilities, the optimal decision is to choose the class that minimizes the expected cost which can be obtained by marginalising the predicted class over all possible true classes [13].

$$\hat{y}(\mathbf{x}) = \operatorname*{arg\,min}_{i=\{1,\dots,K\}} \mathbb{E}_{j\sim P(\cdot|\mathbf{x})}(c_{i|j}) = \operatorname*{arg\,min}_{i=\{1,\dots,K\}} \sum_{j=1}^{K} P(C_j|\mathbf{x})c_{i|j}. \tag{3.2}$$

In the binary case we want to predict class C_1 if and only if predicting class C_1 has a lower expected cost than predicting class C_2

$$\begin{align}
\sum_{j=1}^{K} P(C_j|\mathbf{x})c_{1|j} &\leq \sum_{j=1}^{K} P(C_j|\mathbf{x})c_{2|j} \tag{3.3}\\
P(C_1|\mathbf{x})c_{1|1} + P(C_2|\mathbf{x})c_{1|2} &\leq P(C_1|\mathbf{x})c_{2|1} + P(C_2|\mathbf{x})c_{2|2} \tag{3.4}\\
& \tag{3.5}
\end{align}$$

with the equality having the same expected cost independent on the predicted class.

$$pc_{1|1} + (1-p)c_{1|2} = pc_{2|1} + (1-p)c_{2|2} \tag{3.6}$$

where $p = P(C_1|\mathbf{x})$.

In the binary classification setting we can derive the optimal threshold t^* of selecting class one if $p \geq t^*$.

$$\begin{align}
t^*c_{1|1} + (1-t^*)c_{1|2} &= t^*c_{2|1} + (1-t^*)c_{2|2} \tag{3.7}\\
(1-t^*)c_{1|2} - (1-t^*)c_{2|2} &= t^*c_{2|1} - t^*c_{1|1} \tag{3.8}\\
(1-t^*)(c_{1|2} - c_{2|2}) &= t^*(c_{2|1} - c_{1|1}) \tag{3.9}\\
(c_{1|2} - c_{2|2}) - t^*(c_{1|2} - c_{2|2}) &= t^*(c_{2|1} - c_{1|1}) \tag{3.10}\\
(c_{1|2} - c_{2|2}) &= t^*(c_{2|1} - c_{1|1}) + t^*(c_{1|2} - c_{2|2}) \tag{3.11}\\
(c_{1|2} - c_{2|2}) &= t^*(c_{2|1} - c_{1|1} + c_{1|2} - c_{2|2}) \tag{3.12}\\
\frac{c_{1|2} - c_{2|2}}{c_{2|1} - c_{1|1} + c_{1|2} - c_{2|2}} &= t^* \tag{3.13}
\end{align}$$

For the previous cost matrix

	Predicted C_1	Predicted C_2
True C_1	0	1
True C_2	1	0

the optimal threshold corresponds to

$$t^* = \frac{c_{1|2} - c_{2|2}}{c_{1|2} - c_{2|2} + c_{2|1} - c_{1|1}} = \frac{1 - 0}{1 + 1 - 0 - 0} = 0.5 \tag{3.14}$$

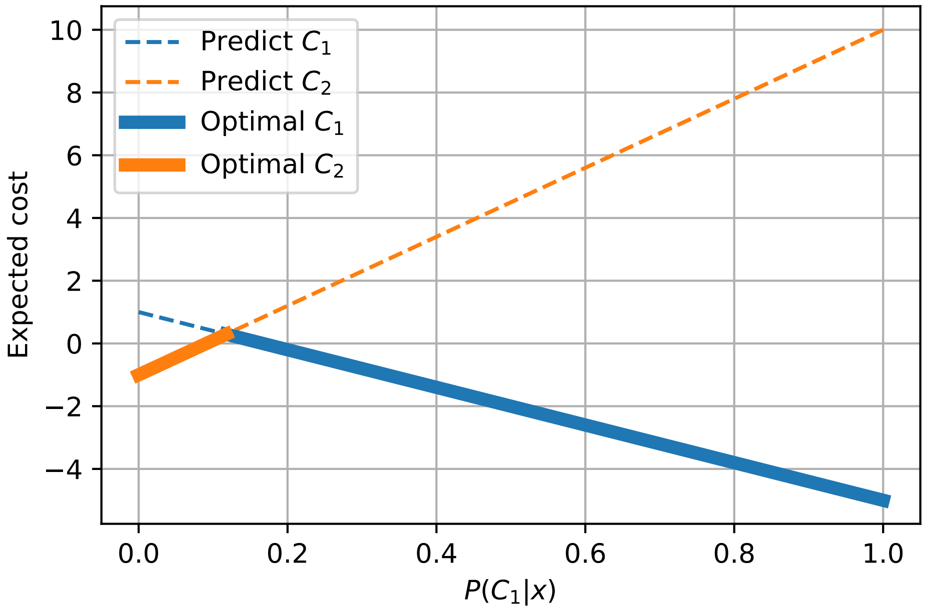
3.4 Different costs binary example

In general, the correct predictions have a cost of 0. However, this may be different in certain scenarios. The following is an example of a cost matrix with different `gains` on the main diagonal and missclassification costs.

	Predicted C_1	Predicted C_2
True C_1	-5	10
True C_2	1	-1

which would result in the following cost lines.

```python
import matplotlib.pyplot as plt

C = [[-5, 1],  # TP, FN
     [10, -1]] # FP, TN
threshold = (C[0][1] - C[1][1])/(C[0][1] - C[1][1] + C[1][0] - C[0][0])
cost_t = threshold*C[0][0] + (1-threshold)*C[0][1]
plt.grid(True)
plt.plot([0, 1], [C[0][1], C[0][0]], '--', label="Predict $C_1$")
plt.plot([0, 1], [C[1][1], C[1][0]], '--', label="Predict $C_2$")
plt.plot([threshold, 1], [cost_t, C[0][0]], lw=5, color='tab:blue', label="Optimal
plt.plot([0, threshold], [C[1][1], cost_t], lw=5, color='tab:orange', label="Optima
plt.xlabel('$P(C_1|x)$')
plt.ylabel('Expected cost')
plt.legend()
plt.show()
```



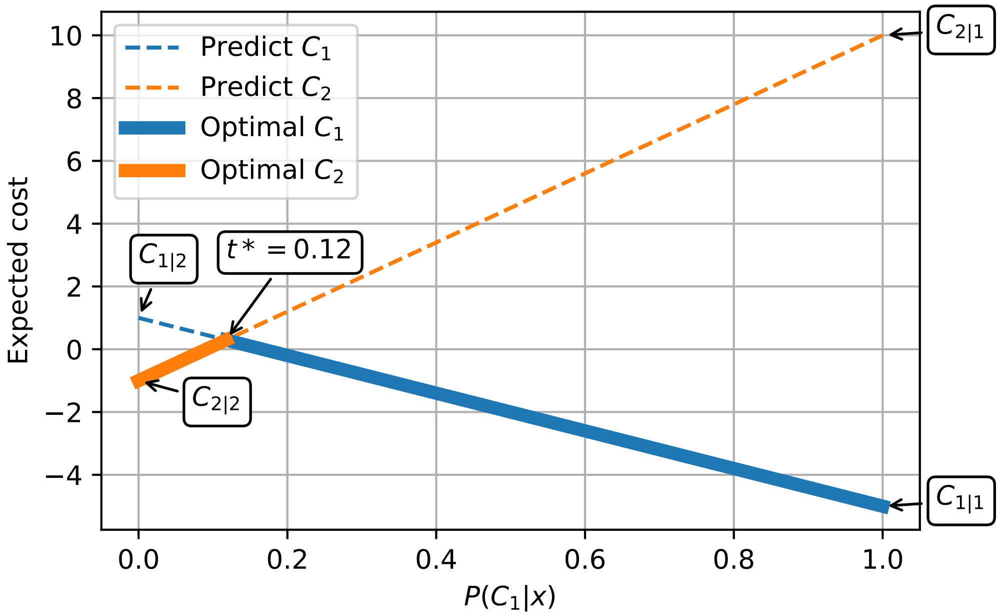
In this case, for a posterior probability vector $[0.4, 0.6]$ we would expect

- Predicting **Class 1** will have an expected cost of $-5 \times 0.4 + 1 \times 0.6 = \mathbf{-1.4}$
- Predicting **Class 2** will have an expected cost of $10 \times 0.4 - 1 \times 0.6 = 3.4$

3.4.1 Other binary examples

```
import matplotlib.pyplot as plt
from shiny import App, render, ui
import pandas as pd

TP = -5
FN = 10
FP = 1
TN = -1
fig = plt.figure()
ax = fig.add_subplot()
ax.grid(True)
ax.plot([0, 1], [FP, TP], '--', label="Predict $C_1$")
ax.plot([0, 1], [TN, FN], '--', label="Predict $C_2$")
```

```
threshold = (FP - TN)/(FP - TN + FN - TP)
cost_t = threshold*TP + (1-threshold)*FP
ax.plot([threshold, 1], [cost_t, TP], lw=5, color='tab:blue', label="Optimal $C_1$"
ax.plot([0, threshold], [TN, cost_t], lw=5, color='tab:orange', label="Optimal $C_2

C = [[TP, FP], [FN, TN]]
bbox = dict(boxstyle="round", fc="white")
ax.annotate(r'$C_{2|2}$', (0, C[1][1]), xytext=(2, -1),
            textcoords='offset fontsize',
            arrowprops=dict(arrowstyle='->', facecolor='black'),
            bbox=bbox)
ax.annotate(r'$C_{1|1}$', (1, C[0][0]), xytext=(2, 0),
            textcoords='offset fontsize',
            arrowprops=dict(arrowstyle='->', facecolor='black'),
            bbox=bbox)
ax.annotate(r'$C_{1|2}$', (0, C[0][1]), xytext=(0, 2),
            textcoords='offset fontsize',
            arrowprops=dict(arrowstyle='->', facecolor='black'),
            bbox=bbox)
ax.annotate(r'$C_{2|1}$', (1, C[1][0]), xytext=(2, 0),
            textcoords='offset fontsize',
            arrowprops=dict(arrowstyle='->', facecolor='black'),
            bbox=bbox)

ax.annotate(f'$t*={threshold:0.2}$', (threshold, cost_t),
            xytext=(0, --3),
            textcoords='offset fontsize',
            arrowprops=dict(arrowstyle='->', facecolor='black'),
            bbox=bbox)

ax.set_xlabel('$P(C_1|x)$')
ax.set_ylabel('Expected cost')
ax.legend()
```



3.5 Cost invariances

The optimal prediction does not change if the cost matrix is

- Multiplied by a positive constant value
- Shifted by a constant value

```
#| standalone: true
#| components: viewer
#| viewerHeight: 480

import numpy as np
import matplotlib.pyplot as plt
from shiny import App, render, ui
import pandas as pd


def fraction_to_float(fraction):
    if '/' in fraction:
       numerator, denominator = fraction.split('/')
       result = float(numerator)/float(denominator)
    else:
       result = float(fraction)
    return result

# X|Y means predict X given that the true label is Y
# Because the indices in a matrix are first row and then column we need to
# invert the order of X and Y by transposing the matrix. Then [0,1] is predict 0
# when the true label is 1.
# TODO: check indices
C_original = np.array([[-2,  3],     # 1|1, 2|1
                       [13, -7]]).T  # 1|2, 2|2

app_ui = ui.page_fluid(
    ui.layout_sidebar(
        ui.panel_sidebar(
            ui.input_slider("S", "Shift constant S", value=0,  min=-10,
                            max=10),
            ui.input_radio_buttons("M", "Multiplicative constant M",
                                   choices=['1/20', '1/10', '1/5', '1',
                                            '5', '10', '20'],
                                   selected = '1', inline=True, width='100%'),
            ui.output_table('cost_matrix'),
        ),
        ui.panel_main(
            ui.output_plot("plot")
        )
    ),
)

def server(input, output, session):
    @output
    @render.plot(alt="A histogram")
    def plot():
        fig = plt.figure()
        ax = fig.add_subplot()
        ax.grid(True)

        global C_original
        C = C_original + input.S()
        C = C*fraction_to_float(input.M())

        threshold = (C[0][1] - C[1][1])/(C[0][1] - C[1][1] + C[1][0] - C[0][0])
        cost_t = threshold*C[0][0] + (1-threshold)*C[0][1]

        ax.plot([0, 1], [C[0][1], C[0][0]], '--', label="Predict $C_1$")
        ax.plot([0, 1], [C[1][1], C[1][0]], '--', label="Predict $C_2$")
        ax.plot([threshold, 1], [cost_t, C[0][0]], lw=5, color='tab:blue', label="Optimal $C_1
        ax.plot([0, threshold], [C[1][1], cost_t], lw=5, color='tab:orange', label="Optimal $C

        bbox = dict(boxstyle="round", fc="white")
        ax.annotate(r'$C_{2|2}$', (0, C[1][1]), xytext=(-0.2, C[1][1]),
                    arrowprops=dict(arrowstyle='->', facecolor='black'),
                    bbox=bbox)
        ax.annotate(r'$C_{1|1}$', (1, C[0][0]), xytext=(1.1, C[0][0]),
                    arrowprops=dict(arrowstyle='->', facecolor='black'),
                    bbox=bbox)
        ax.annotate(r'$C_{1|2}$', (0, C[0][1]), xytext=(-0.2, C[0][1]),
                    arrowprops=dict(arrowstyle='->', facecolor='black'),
                    bbox=bbox)
        ax.annotate(r'$C_{2|1}$', (1, C[1][0]), xytext=(1.1, C[1][0]),
                    arrowprops=dict(arrowstyle='->', facecolor='black'),
                    bbox=bbox)

        ax.annotate(f'$t*={threshold:0.2}$', (threshold, cost_t),
                    xytext=(threshold + 0.2, cost_t),
                    arrowprops=dict(arrowstyle='->', facecolor='black'),
                    bbox=bbox)

        ax.set_xlabel('$P(C_1|x)$')
        ax.set_ylabel('Expected cost')
        ax.legend()

        return fig
    @output
    @render.table(index=True)
    def cost_matrix():
        global C_original
        C = C_original.T + input.S() # Need to transpose back to show print matrix
        C = C*fraction_to_float(input.M())

        return pd.DataFrame(C,
                            index=['True C1', 'True C2'],
                            columns=['Predicted C1', 'Predicted C2'])

app = App(app_ui, server, debug=True)
```

3.5.1 Simplification example

Because of these invariances, it is common in the binary case to modify the matrix c in such a way that the missclassification cost for one of the classes is 1 and a cost of 0 for its correct prediction. For example, if $c^*_{1|2} = 1$ and $c^*_{2|2} = 0$ we get

$$t^* = \frac{c_{1|2} - c_{2|2}}{c_{1|2} - c_{2|2} + c_{2|1} - c_{1|1}} = \frac{1}{1 + c^*_{2|1} - c^*_{1|1}} \tag{3.15}$$

In the previous example the original cost matrix c

$$c = \begin{bmatrix} -2 & 3 \\ 13 & -7 \end{bmatrix}^\top \tag{3.16}$$

if shifted by +7 and scaled by 1/20 results in

$$c' = \begin{bmatrix} (-2+7)/20 & (3+7)/20 \\ (13+7)/20 & (-7+7)/20 \end{bmatrix}^\top = \begin{bmatrix} 0.25 & 0.5 \\ 1 & 0 \end{bmatrix}^\top \tag{3.17}$$

with an optimal threshold

$$t^* = \frac{1}{1 + c_{2|1}{}' - c'_{1|1}} = \frac{1}{1 + 0.5 - 0.25} = 0.8 \tag{3.18}$$

3.6 Multiclass setting

The binary cost matrix can be extended to multiclass by extending the rows with additional true classes and columns with predicted classes.

	Predicted C_1	Predicted C_2	$\cdots$	Predicted C_K
True C_1	$c_{1\|1}$	$c_{2\|1}$	$\cdots$	$c_{K\|1}$
True C_2	$c_{1\|2}$	$c_{2\|2}$	$\cdots$	$c_{2\|2}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
True C_K	$c_{1\|K}$	$c_{2\|K}$	$\cdots$	$c_{K\|K}$

However, with more than 2 classes the threshold is not a single value but multiple decision boundaries in the probability simplex.

3.7 Ternary example {.smaller}

In order to exemplify the process of making an optimal decision in more with more than two classes we can look at the ternary case, which naturally extends to more classes. Given the following cost matrix

	Predicted C_1	Predicted C_2	Predicted C_3
True C_1	-10	20	30
True C_2	40	-50	60
True C_3	70	80	-90

and a true posterior probability vector for all the classes $[0.5, 0.1, 0.4]$, we can estimate the expected cost of making each class prediction

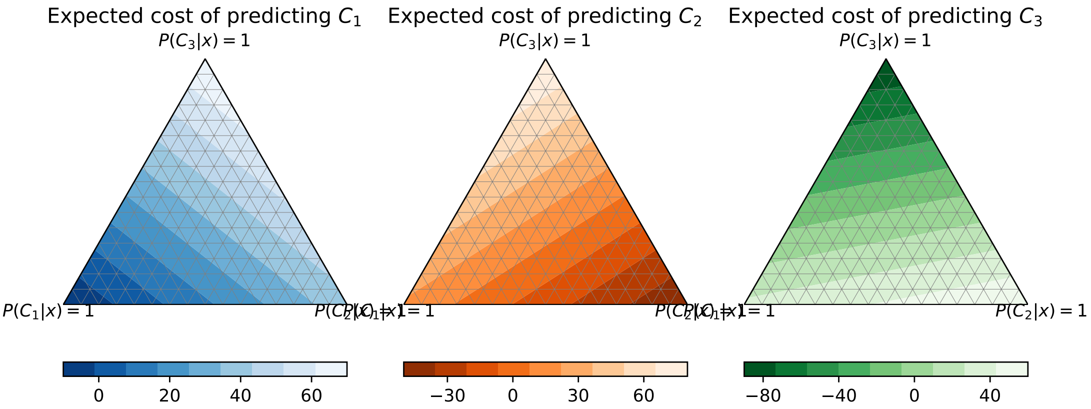
$$\mathbb{E}_{j\sim P(\cdot|\mathbf{x})}(c_{i|j}) = \sum_{j=1}^{K} P(C_j|\mathbf{x})c_{i|j}. \tag{3.19}$$

which results in the following expected costs:

- Predicting **Class 1** will have a cost of $-10 \times 0.5 + 40 \times 0.1 + 70 \times 0.4 = 27$
- Predicting **Class 2** will have a cost of $20 \times 0.5 - 50 \times 0.1 + 80 \times 0.4 = 37$
- Predicting **Class 3** will have a cost of $30 \times 0.5 + 60 \times 0.1 - 90 \times 0.4 = \mathbf{-15}$

3.7.1 Ternary expected cost isolines per decision

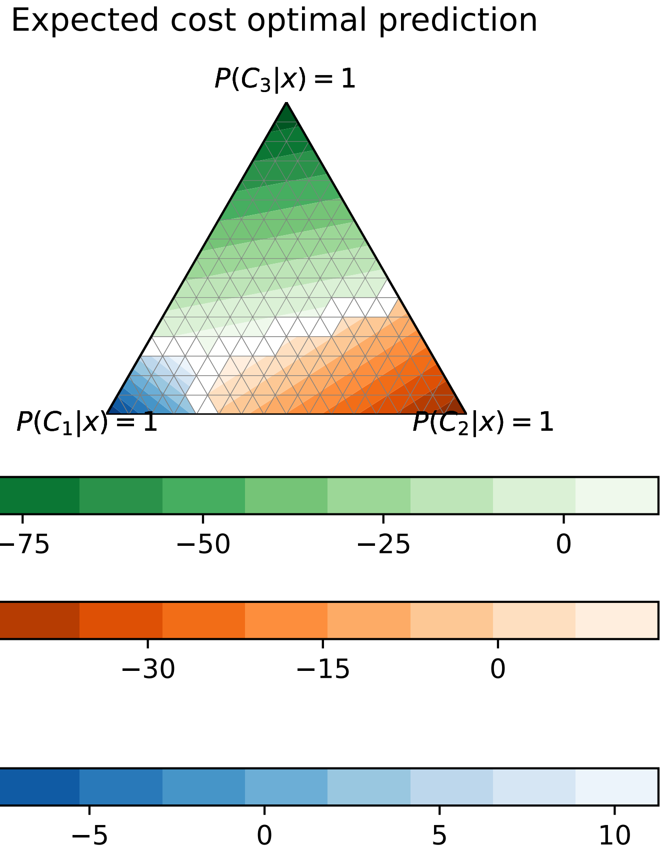
```
import matplotlib.pyplot as plt
from pycalib.visualisations.barycentric import draw_func_contours

C = [[-10, 40, 70], [20, -50, 80], [30, 60, -90]]

cmaps = ['Blues_r', 'Oranges_r', 'Greens_r']
labels = [f"$P(C_{i+1}|x) = 1$" for i in range(len(C))]

fig = plt.figure(figsize=(10, 4))
for i in range(len(C)):
    ax = fig.add_subplot(1, len(C), i+1)
    def cost_func(prob):
        return sum(prob*C[i])

    ax.set_title(f"Expected cost of predicting $C_{i+1}$\n")
    draw_func_contours(cost_func, labels=labels, nlevels=10, subdiv=4,
                       cmap=cmaps[i], fig=fig, ax=ax)
```



3.7.2 Ternary hyperplanes optimal decision combined

```
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
from pycalib.visualisations.barycentric import draw_func_contours

C = [[-10, 40, 70], [20, -50, 80], [30, 60, -90]]

cmaps = ['Blues_r', 'Oranges_r', 'Greens_r']
labels = [f"$P(C_{i+1}|x) = 1$" for i in range(len(C))]

fig = plt.figure(figsize=(5, 5))
ax = fig.add_subplot()
fig.suptitle(f"Expected cost optimal prediction")
for i in range(len(C)):
    def cost_func(prob):
        expected_costs = np.inner(prob, C)
        min_p_id = np.argmin(expected_costs)
        if min_p_id == i:
            return expected_costs[i]
        return np.nan

    draw_func_contours(cost_func, labels=labels, nlevels=10, subdiv=4,
                       cmap=cmaps[i], cb_orientation='vertical', fig=fig, ax=ax)


plt.show()
```

```
/opt/hostedtoolcache/Python/3.10.14/x64/lib/python3.10/site-packages/pycalib/visualisations/ba

The following kwargs were not used by contour: 'cb_orientation'
```



3.8 Option to abstain

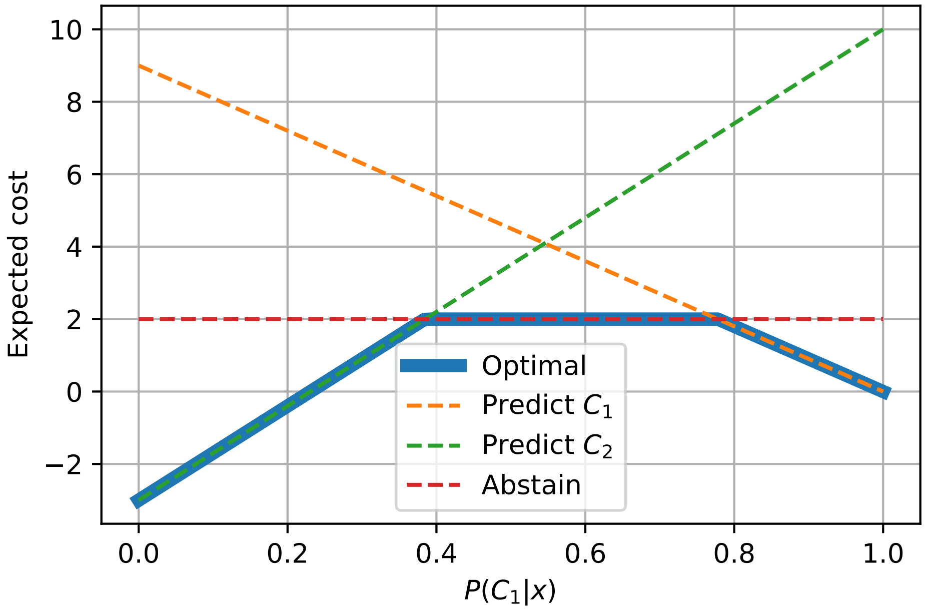
It is possible to add the costs of abstaining on making a prediction by adding a column into the original cost matrix [4]. The following is an example which illustrates this in a binary classification problem.

	Predicted C_1	Predicted C_2	Abstain
True C_1	0	10	2
True C_2	9	-3	2

- Predicting **Class 1** has an expected cost of $0 \times 0.3 + 9 \times 0.7 = 6.3$
- Predicting **Class 2** has an expected cost of $10 \times 0.3 - 3 \times 0.7 = \mathbf{0.9}$
- **Abstaining** has an expected cost of $2 \times 0.3 + 2 \times 0.7 = 2$

3.8.1 Option to abstain cost lines

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

C = [[0, 9], [10, -3], [2, 2]]
p = np.linspace(0, 1, 100)
p = np.vstack([1 - p, p]).T
opt_cost = [min(np.inner(C, p[i])) for i in range(p.shape[0])]
plt.plot(p[:,0], opt_cost, lw=5, label='Optimal')

plt.grid(True)
plt.plot([0, 1], [C[0][1], C[0][0]], '--', label="Predict $C_1$")
plt.plot([0, 1], [C[1][1], C[1][0]], '--', label="Predict $C_2$")
plt.plot([0, 1], [C[2][1], C[2][0]], '--', c='tab:red', label="Abstain")
plt.xlabel('$P(C_1|x)$')
plt.ylabel('Expected cost')
plt.legend()
plt.show()
```

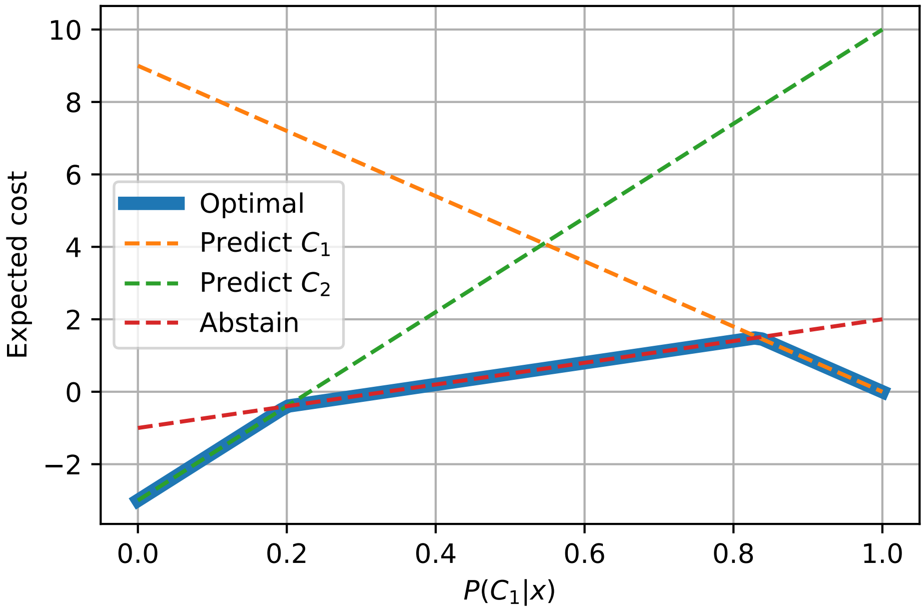


3.8.2 Option to abstain different costs

The following is another example in which abstaining from making a prediction if the true class was C_2 would incur into a `gain`.

	Predicted C_1	Predicted C_2	Abstain
True C_1	0	10	2
True C_2	9	-3	-1

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

C = np.array([[0, 9], [10, -3], [2, -1]])
p = np.linspace(0, 1, 100)
p = np.vstack([1 - p, p]).T
opt_cost = [min(np.inner(C, p[i])) for i in range(p.shape[0])]
plt.plot(p[:,0], opt_cost, lw=5, label='Optimal')

plt.grid(True)
plt.plot([0, 1], [C[0][1], C[0][0]], '--', label="Predict $C_1$")
plt.plot([0, 1], [C[1][1], C[1][0]], '--', label="Predict $C_2$")
plt.plot([0, 1], [C[2][1], C[2][0]], '--', c='tab:red', label="Abstain")
plt.xlabel('$P(C_1|x)$')
plt.ylabel('Expected cost')
plt.legend()
plt.show()
```



References

[1] Reem Alotaibi and Peter Flach. “Multi-label thresholding for cost-sensitive classification”. In: *Neurocomputing* 436 (May 2021), pp. 232–247. ISSN: 0925-2312. DOI: 10.1016/J.NEUCOM.2020.12.004.

[2] Edmon Begoli, Tanmoy Bhattacharya, and Dimitri Kusnezov. “The need for uncertainty quantification in machine-assisted medical decision making”. In: *Nature Machine Intelligence* 1.1 (2019), pp. 20–23.

[3] Klaus Brinker and Eyke Hüllermeier. “A reduction of label ranking to multiclass classification”. In: *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part III.* Springer. 2020, pp. 204–219.

[4] Nontawat Charoenphakdee et al. “Classification with Rejection Based on Cost-sensitive Classification”. In: ed. by Marina Meila and Tong Zhang. Vol. 139. - summary: The authors propose a very simple method to perform multi-class classification with reject option without the need of estimating accurate posterior probabilities. The reason being that the estimation of probabilities is more difficult, and not necessary in the context with fixed costs. The basic idea is to train K one-vs-rest classifiers with a zero-one-c loss (zero for correct classification, one for incorrect classification and c for abstention), then abstain if none of the models predict their class (uncertainty), or if more than one does (because of ambiguity), and predict the winning class otherwise. Their method only requires classification-calibrated losses to train and the results look very good for their method with sigmoid loss even with noisy data. One of the compared methods confidence-based (to estimate accurate posterior probabilities) is softmax with cross-entropy loss and Temperature scaling.- Proposes a new method to perform classification with a reject option by connecting hte cost-sensitive classification with classification with reject option.- Based on an ensemble of cost sensitive classifiers- (1) Do not require estimating posterior probabilities- (2) flexible losses even if non-convex- (3) easy ensemble with different losses- (4) binary or multiclass- (5) theory to support for any classification-calibrated loss- Cost-based framework(Chow, 1970; Barlett and Wegkamp, 2008; Yuan and Wegkamp, 2010; Cortes et al., 2016; Frank and Prusa, 2019; Ni et al., 2019) is commonly ussed for this setting- (A) confidence-based approaches (Bartlett and Wegkamp, 2008; Crandvalet et al. 2009; Herbei and Wegkamp, 2006; Yuan and Wegkamp, 2010; Ramaswamy et al., 2018; Ni et al., 2019): train posterior probability estimators and uses the confidence values to make the decision, which usually require proper losses, and may have difficulties

to properly estimate the probabilities.- Some exceptions that do not require posterior probability estimations only for binary problems exist- (B) classifier-rejector (Cortes et al 2016): train a classifier and a rejector separately. It is theoreticall justified for binary case with hinge-loss. Does not seem applicable to the multiclass case with theoretical justification and empirical evidence (Ni et al 2019).- The main point is that the Bayes optimal solution only requires knowing which class has the maximum posterior probability, and if the posterior probability of that class is higher than 1 minus the cost c. (I need to understand this).- MPN: One think to bear in mind is that once this model is learned, I do not think it can be adjusted to new operating conditions. Need to check this.- They assume the rejection cost c is between 0 and 0.5- The authors use the zero-one-c loss (Ni et al., 2019) which is a zero-one loss with cost c when rejection.- Information for the case in which c > 0.5 can be found in Ramaswamy et al. (2018)- Chow's rule requires the posterior probabilities, if the confidence (maximum posterior among classes) is smaller than 1 - c then the model rejects. Otherwise the class with maximum posterior is chosen- Cost-sensitive binary does not assume equal costs. The authors propose alpha for one and 1 - alpha for the other. Scott (2012) defines the optimal cost-sensitive classifier predicting positive if its probability is higher than alpha, and negative otherwise.- The author explains that in Chow's rule, once the c is known, we do not need the posterior probabilities, but to know that this is below or above the threshold c. Which simplifies the overall task to classification rather than probability estimation (as suggested by Vapnik, 1998). - The multiclass case is also be done with Chow's rule and K binary cost-sensitive classifiers, when all of them reject, then reject, if not, select the class with maximum prediction.- A binary margin surrogate loss is proposed in Definition 5.- The final classification rule rejects if all binary predictions are negative, or si more than one is positive (ambiguity). Otherwise the only positive class is predicted.- Compared methods: (SCE) the confidence-based method with Softmax cross-entropy, (DEFER) the classifier-reject method proposed in Mozannar and Sontag (2020), (ANGLE) the beng hinge loss by Zhang et al. (2018), (CS-hinge) the authors method with hinge loss and (CS-sigmoid) with sigmoid loss.- Results for binary problems with clean and noisy datasets, as well as positive-unlabeled dataset shows generally the smallest zero-one-c risk for the proposed method with sigmoid loss (CS-sigmoid).- Their method with hinge loss got small risk on clean dataset and positive-ulabelled, but not in noisy settings.- The risk on the compared methods had mixed results.- Results on the multiclass setting show their method CS-sigmoid performing generally well (and the best in multiple ocasions), CS-hinge in a noisy setting performs bad, and sigmoid with cross entropy and temperature scaling (SCE) performed performed very well ocasionally. PMLR, Oct. 2021, pp. 1507–1517. URL: https://proceedings.mlr.press/v139/charoenphakdee21a.html%20http://arxiv.org/abs/2010.11748.

[5] Lize Coenen, Ahmed KA Abdullah, and Tias Guns. “Probability of default estimation, with a reject option”. In: *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE. 2020, pp. 439–448.

[6] Chris Drummond and Robert C. Holte. “Cost curves: An improved method for visualizing classifier performance”. In: *Machine Learning* 65.1 (2006), pp. 95–130.

[7] Didier Dubois and Henri Prade. “Possibility theory, probability theory and multiple-valued logics: A clarification”. In: *Annals of mathematics and Artificial Intelligence* 32 (2001), pp. 35–66.

[8] Charles Elkan. “The Foundations of Cost-Sensitive Learning The Foundations of Cost-Sensitive Learning”. In: *17th International Conference on Artificial Intelligence (IJCAI’01)*. Ed. by Morgan Kaufmann. May 2001. 2001, pp. 973–978.

[9] Isaac Levi. *The enterprise of knowledge: An essay on knowledge, credal probability, and chance*. MIT press, 1980.

[10] Hussein Mozannar and David Sontag. “Consistent estimators for learning to defer to an expert”. In: ed. by Hal Daumé III and Aarti Singh. Vol. PartF16814. PMLR, 2020, pp. 7033–7044. ISBN: 9781713821120. URL: https://proceedings.mlr.press/v119/mozannar20b.html.

[11] Galen E Mullins et al. “Adaptive generation of challenging scenarios for testing and evaluation of autonomous vehicles”. In: *Journal of Systems and Software* 137 (2018), pp. 197–215.

[12] Isaac Kofi Nti, Adebayo Felix Adekoya, and Benjamin Asubam Weyori. “A systematic review of fundamental and technical analysis of stock market predictions”. In: *Artificial Intelligence Review* 53.4 (2020), pp. 3007–3057.

[13] Deirdre B O’Brien, Maya R Gupta, and Robert M Gray. “Cost-Sensitive Multi-Class Classification from Probability Estimates”. In: Association for Computing Machinery, 2008, pp. 712–719. ISBN: 9781605582054. DOI: 10.1145/1390156.1390246. URL: https://doi.org/10.1145/1390156.1390246.

[14] Adnan Qayyum et al. “Securing connected & autonomous vehicles: Challenges posed by adversarial machine learning and the way forward”. In: *IEEE Communications Surveys & Tutorials* 22.2 (2020), pp. 998–1026.

[15] Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. *Algorithmic learning in a random world*. Vol. 29. Springer, 2005.

[16] Qian Yang, Aaron Steinfeld, and John Zimmerman. “Unremarkable AI: Fitting intelligent decision support into critical, clinical decision-making processes”. In: *Proceedings of the 2019 CHI conference on human factors in computing systems*. 2019, pp. 1–11.

[17] Bianca Zadrozny and Charles Elkan. “Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers”. In: 2001, pp. 609–616. ISBN: 1-55860-778-1. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.29.3039&rep=rep1&type=pdf.