# CS342 Project: Bayes Classifiers, Calibration, and Multicalibration

In this project, you will first learn about *calibration*, and you will then explore this concept in the context of image classification models. This project will help you understand the distinction between accuracy and reliability, and how calibration and multicalibration can be used to assess fairness and model trustworthiness.

## 1 Background and Motivation

Machine learning models often output *probabilities*. For example, a model might predict that a person in an image is smiling with probability 0.8. A natural question arises: *Can we trust these probabilities?* If the model assigns 0.8 to many images and about 80% of those are truly smiling, the model is said to be **well calibrated**. If only 60% are smiling, it is **overconfident**; if 95% are smiling, it is **underconfident**. A classifier can be highly accurate but poorly calibrated. While we have focused on prediction accuracy in most of this module, calibration is also of fundamental concern, and it matters when probabilities are used to make downstream decisions, e.g., when deciding whether to show a warning, request human review, or combine multiple models.

We consider prediction tasks over a domain $\mathcal{X} \times \mathcal{Y}$. Here, $\mathcal{X}$ is the feature domain and $\mathcal{Y} = \{0, 1\}$ is the label domain. We assume examples are sampled from a distribution $\mathcal{D}$ over $\mathcal{Z}$. Given such a distribution, we use $\mathcal{D}_{\mathcal{X}}$ to denote the marginal distribution over $\mathcal{X}$. For a particular feature vector $\mathbf{x}$, let $\mathcal{D}_{\mathcal{Y}}(\mathbf{x})$ denote the conditional distribution on $\mathcal{Y}$, given that the feature vector is $\mathbf{x}$; this is the distribution we are trying to estimate. A *model* is a function $f : \mathcal{X} \to [0, 1]$, and our goal is to find a model $f^*$ such that for every $\mathbf{x}$:

$$f^*(\mathbf{x}) = \Pr_{y \sim \mathcal{D}_{\mathcal{Y}}(\mathbf{x})}[y = 1].$$

If our goal is purely prediction, we might evaluate a model $f$ via its *mean squared error (MSE)*:

$$\mathrm{MSE}(f) = \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}}[(f(\mathbf{x}) - y)^2].$$

Calibration asks that the predictions of $f$ be correct conditional on its own predictions. We quantify calibration using the metric of *expected calibration error (ECE)*:

$$\mathrm{ECE}(f) = \sum_{v \in R(f)} \Pr_{\mathbf{x}\sim\mathcal{D}_{\mathcal{X}}}[f(\mathbf{x}) = v] \cdot \left| v - \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}}[y \mid f(\mathbf{x}) = v] \right|,$$

where $R(f)$ is the range of $f$. Typically, we will ensure that $|R(f)| \leq 10$; if $|R(f)| > 10$, we can "bin" values of $f$ in interval $[0, 0.1)$ to 0, values in interval $[0, 1, 0.2)$ to 0.1, values in interval $[0, 2, 0.3)$ to 0.2, etc.

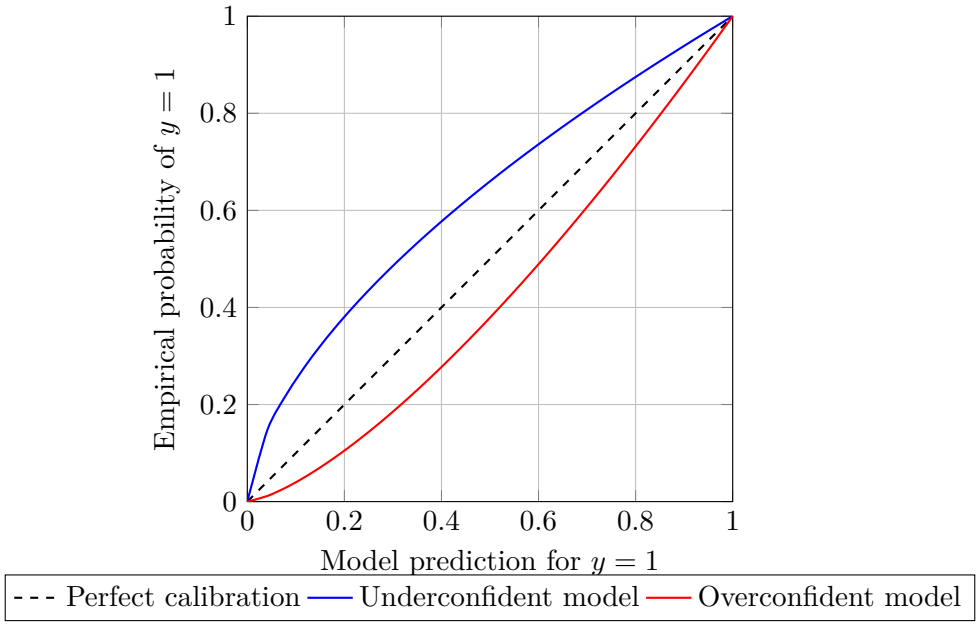
MSE and ECE cannot be computed exactly, as we only have access to samples from $\mathcal{D}$. We instead resort to computing their empirical versions, $\widehat{\mathrm{MSE}}$ and $\widehat{\mathrm{ECE}}$. Given $n$ samples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ from $\mathcal{D}$, define:

$$\widehat{\mathrm{MSE}}(f) = \frac{1}{n} \sum_i (f(\mathbf{x}_i) - y_i)^2;$$

$$\widehat{\mathrm{ECE}}(f) = \sum_{v \in R(f)} \frac{n_v}{n} \left| v - \frac{\sum_{i:f(\mathbf{x}_i)=v} y_i}{n_v} \right|,$$

where $n_v$ is the number of samples such that $f(\mathbf{x}_i) = v$.

Calibration can also be visualised using a *reliability diagram*, which plots $v$ versus $\frac{\sum_{i:f(\mathbf{x}_i)=v} y_i}{n_v}$. A perfectly calibrated model gives a curve that lies along the diagonal $y = x$. Curves above this line describe underconfident models, while those below this describe overconfident models.




A model may be well calibrated overall but miscalibrated for certain subsets or groups (e.g. systematically overconfident for male faces). Population calibration does not guarantee group calibration, a key consideration for fairness and trustworthiness. *Multicalibration* requires that a model be well calibrated for every group in a collection of groups. Let $\mathcal{G}$ denote a collection of groups, i.e., subsets of the feature domain $\mathcal{X}$. We will say that a model $f$ is *$\varepsilon$-multicalibrated* with respect to $\mathcal{G}$ if for every group $g \in \mathcal{G}$,

$$\mathrm{ECE}(f; g) = \sum_{v \in R(f)} \Pr_{\mathbf{x} \sim \mathcal{D}_{\mathcal{X}}} [f(\mathbf{x}) = v \mid \mathbf{x} \in g] \cdot \left| v - \mathop{\mathbb{E}}_{(\mathbf{x},y) \sim \mathcal{D}} [y \mid f(\mathbf{x}) = v, \mathbf{x} \in g] \right|$$

is at most $\varepsilon$. In practice, the conditional probabilities and expectations in $\mathrm{ECE}(f; g)$ would be estimated using their empirical versions. Also, this definition assumes that for each group $g$, there is significant probability of drawing $\mathbf{x}$ that lies in $g$, so that one can estimate $\mathrm{ECE}(f; g)$ from samples; this will be true in your experiments.

## 2 Project Overview

You will use the **CelebA** dataset of celebrity faces. Each image has 40 binary attributes such as *Smiling*, *Male*, *Young*, *Blond_Hair*, and *Wearing_Hat*. Your goal is to predict the *Smiling* attribute from the image using features extracted from a pre-trained Vision Transformer (ViT), then evaluate and improve the calibration of predicted probabilities.

# 3 Project Tasks

### Part A – Feature Extraction and Bayes Classifier

1. **Feature extraction:** Use a pre-trained Vision Transformer (`vit_b_16` from `torchvision.models`) as a fixed feature extractor. Represent each image by its embedding vector (about 768 dimensions). Use a subset of 20,000 images. Partition the set into 70% for training, 15% for calibration, and 15% for testing. Train on the training set (70%), fit calibration methods on the calibration set (15%), and evaluate all metrics on the test set (15%).
2. **Train classifiers:** Implement a Naive Bayes classifier to predict the *Smiling* attribute (1 = smiling, 0 = not smiling) and report test accuracy. The Naive Bayes classifier should be **implemented by you** and not be a call to a library function. The features here are numerical (not categorical). So conditioned on the label, model each feature as an independent Gaussian. Your algorithm should estimate the mean and variance of each of these Gaussians, using the training data set.

### Part B – Population Calibration

1. Compute predicted probabilities $\hat{p}(x)$ for all samples in your calibration set, and plot a reliability diagram: bin predictions and compare average predicted probability with actual fraction of smiling faces.
2. Compute the ECE and MSE scores on the test set. Comment on whether the model is overconfident or underconfident.

### Part C – Subgroup Calibration

Use the following 8 subgroups from CelebA attributes:

- Gender: *Male / Female*
- Age: *Young / Not Young*
- Hair colour: *Blond_Hair / Not Blond_Hair*
- Accessory: *Wearing_Hat / Not Wearing_Hat*

1. For each subgroup, plot a reliability diagram and compute ECE and MSE score. Summarise subgroup size, accuracy, ECE, and MSE in a table.
2. Discuss which subgroups are least well calibrated and whether overall calibration conceals subgroup differences.

### Part D – Improving Calibration and Multicalibration

This part explores how post-hoc calibration affects subgroup reliability and introduces a simple form of multicalibration.

1. Fit parameters $a, b$ on the calibration set to minimise mean squared error:

$$\min_{a,b} \sum_{v \in R(f)} \left( av + b - \frac{\sum_{i:f(\mathbf{x}_i)=v} y_i}{n_v} \right)^2.$$

For prediction, clip the outputs to $[0, 1]$:

$$\hat{p}(x) = \min(\max(a \cdot f(x) + b, 0), 1).$$

Implement the linear regression routine yourself (not a library call). Plot this fitted line against the empirical reliability curve.

For comparison, also apply two standard calibration methods:

- **Platt scaling:** logistic mapping $\sigma(As + B)$. Use the following code[1]:

```
# 1. Platt Scaling (Sigmoid)
# We use cv="FrozenEstimator" because gnb is already trained on X_train.
# We fit the calibrator on the separate X_cal, y_cal set.
cal_gnb_platt = CalibratedClassifierCV(gnb, method='sigmoid',
      cv="FrozenEstimator")
cal_gnb_platt.fit(X_cal, y_cal)

# Get new probabilities on the test set
gnb_probs_platt = cal_gnb_platt.predict_proba(X_test)[:, 1]
calibrated_models['GNB_Platt'] = cal_gnb_platt
calibrated_probs['GNB_Platt'] = gnb_probs_platt
```

- **Isotonic regression:** non-decreasing stepwise mapping. Similar to above but use

```
cal_gnb_iso = CalibratedClassifierCV(gnb, method='isotonic', cv="FrozenEstimator")
```

Compare all three methods (linear, Platt, isotonic) in terms of population ECE and MSE on the test set.

Using the same subgroups as in Part C, recompute accuracy, ECE, and MSE scores. Analyse whether global calibration improved or worsened subgroup reliability. Does improving population calibration automatically improve subgroup calibration?

2. To move toward multicalibration, implement an iterative correction procedure.

**Algorithm.**

1. **Initial model:** start from the globally calibrated model.
2. **Groups:** use the eight groups
   - Male / Female
   - Young / Not Young
   - Blond_Hair / Not Blond_Hair
   - Wearing_Hat / Not Wearing_Hat
3. **Iterate for up to five rounds:**
   3.1. Compute ECE($G$) for each group $G$.
   3.2. Identify the worst-calibrated group $G^* = \arg\max_G \text{ECE}(G)$.

[1] If your scikit-learn version does not support cv="prefit", you can instead wrap the trained classifier using CalibratedClassifierCV with cv='prefit'.

3.3. Recalibrate only on data from $G^*$ (using Platt or isotonic) to obtain updated probabilities.

3.4. Replace the model's predictions for samples in $G^*$ with their recalibrated values.

4. Stop when all groups have ECE$<0.03$ or after five iterations.

Record the worst-group ECE and population ECE after each iteration. Plot their evolution and compare the final multicalibrated model with the globally calibrated one.

# 4 Deliverables

1. **Jupyter notebook (`project.ipynb`)** with full code that is runnable.

2. **Summary report (`summary.pdf`)**, answering questions above and a 100-word description of calibration in your own words.

3. **1-2-minute video (`overview.mp4`)**, showing both yourself and your screen[2], where you narrate the structure of your code and a quick description of your results.

# Appendix: Implementation Hints and Practical Guidance

## A. Dataset preparation

- Download CelebA from the official CelebA website or a hosted mirror.
- Each image has a corresponding row in `list_attr_celeba.txt` containing 40 binary attributes.
- Focus on the attributes: `Smiling`, `Male`, `Young`, `Blond_Hair`, and `Wearing_Hat`.
- Randomly sample ∼20,000 images to keep training fast. Store their attribute rows for later use.

## B. Vision Transformer feature extraction

You will use a pre-trained Vision Transformer (ViT) to obtain image embeddings that serve as features for your Bayes and logistic regression classifiers. The example below shows how to extract and save embeddings using `torchvision`.

```
import torch
from torchvision import models, transforms
from PIL import Image
import numpy as np
from tqdm import tqdm
import os
```

[2]You should be able to create such a video by entering a meeting with yourself on Microsoft Teams, sharing your screen, and recording the session

```
def extract_features():
    """
    Extracts features from images using a pre-trained Vision Transformer (ViT)
    and saves them to a file.
    """
    if os.path.exists(EMBEDDINGS_FILE):
        print(f"Embeddings file '{EMBEDDINGS_FILE}' already exists. Skipping feature extract
        return

    print("Starting feature extraction with Vision Transformer...")
    # 1. Load pre-trained Vision Transformer
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    vit = models.vit_b_16(weights=models.ViT_B_16_Weights.IMAGENET1K_V1).to(device)
    vit.eval()  # Set model to evaluation mode

    # 2. Define preprocessing steps consistent with ImageNet training
    preprocess = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ])

    # 3. Get the list of images to process
    image_list = sorted(os.listdir(IMAGE_DIR))[:NUM_IMAGES_TO_PROCESS]
    all_features = []

    # 4. Extract embeddings for each image
    with torch.no_grad():
        for fname in tqdm(image_list, desc="Extracting ViT Embeddings"):
                img = Image.open(os.path.join(IMAGE_DIR, fname)).convert("RGB")
                x = preprocess(img).unsqueeze(0).to(device)

                # Manually replicate the forward pass to get the features before the
                # classification head, as the internal API (like .process_input)
                # can change.

                # 1. Process input using the private _process_input method
                x_processed = vit._process_input(x)
                n = x_processed.shape[0]

                # 2. Add the class token
                batch_class_token = vit.class_token.expand(n, -1, -1)
                x_with_token = torch.cat([batch_class_token, x_processed], dim=1)

                # 3. Pass through the encoder
                encoded_features = vit.encoder(x_with_token)

                # 4. Get the class token's output (this is the feature vector)
                features = encoded_features[:, 0]
```

```
                all_features.append(features.cpu().numpy().flatten())


    all_features_np = np.array(all_features)
    print(f"Feature matrix shape: {all_features_np.shape}")

    # 5. Save embeddings for later use
    np.save(EMBEDDINGS_FILE, all_features_np)
    print(f"Embeddings saved to '{EMBEDDINGS_FILE}'.")
```

### C. Data splits

Use three disjoint sets:

- **Training set:** for fitting classifiers (70%)
- **Calibration set:** for calibration (15%)
- **Test set:** for final evaluation (15%)

**Important:** do not use the test set when fitting calibration models; otherwise, ECE will be overly optimistic.

### D. Practical runtime tips

- Use CPU-only execution if GPUs are unavailable; extracting embeddings for 20k images takes roughly 15–30 minutes.
- Cache embeddings to avoid recomputation.
- Limit subgroup and multicalibration iterations if runtime is long; 5 iterations is sufficient.
- Use `matplotlib` to produce clear reliability diagrams with 45° reference lines.