# CS 589: Homework 05

Vishnu Vardhan Reddy B.

October 30, 2025

## Question 1: Compute evaluation metrics for classification.

### a. Class proportions

Total number of samples: $52 + 1 + 3 + 6 + 13 + 7 + 2 + 4 + 12 = 100$
True Flu: $52 + 1 + 3 = 56$
True COVID: $6 + 13 + 7 = 26$
True Pneumonia: $2 + 4 + 12 = 18$

Class proportions:
Flu: $56/100 = 0.56$
COVID: $26/100 = 0.26$
Pneumonia: $18/100 = 0.18$

### b. Error Rate

Correct predictions: $52 + 13 + 12 = 77$
Incorrect predictions: $100 - 77 = 23$
Overall Error Rate: $23/100 = 0.23$ or $23\%$

### c. Balanced Error Rate

Per-class error rates:
Flu Error Rate: $(1 + 3)/56 = 4/56 \approx 0.0714$
COVID Error Rate: $(6 + 7)/26 = 13/26 = 0.5000$
Pneumonia Error Rate: $(2 + 4)/18 = 6/18 \approx 0.3333$

Balanced Error Rate (BER): $(4/56 + 13/26 + 6/18)/3 \approx (0.0714 + 0.5000 + 0.3333)/3 \approx 0.9047/3 \approx 0.3016$

### d. Precision (COVID as positive)

True Positives (TP) for COVID: 13
False Positives (FP) for COVID: Predictions of COVID when true was Flu or Pneumonia = 1+4 = 5
Precision = TP / (TP + FP) = $13/(13 + 5) = 13/18 \approx 0.7222$

### e. Recall (COVID as positive)

True Positives (TP) for COVID: 13
False Negatives (FN) for COVID: True COVID predicted as Flu or Pneumonia = $6 + 7 = 13$
Recall = TP / (TP + FN) = $13/(13 + 13) = 13/26 = 0.5000$

### f. F1 Score (COVID as positive)

Precision (P) = $13/18$
Recall (R) = $13/26 = 0.5$
F1 Score = $2 \times (P \times R)/(P + R) = 2 \times (13/18 \times 0.5)/(13/18 + 0.5) = 2 \times (13/36)/(13/18 + 9/18) = (26/36)/(22/18) = (13/18)/(11/9) = (13/18) \times (9/11) = 13/22 \approx 0.5909$

### g. Misclassification Cost

Cost = (Pred Flu — True COVID) * Count + (Pred Pneumonia — True COVID) * Count + (Pred COVID — True Flu) * Count + (Pred Pneumonia — True Flu) * Count + (Pred Flu — True Pneumonia) * Count + (Pred COVID — True Pneumonia) * Count

Cost = $(3000 \times 6) + (10000 \times 7) + (1500 \times 1) + (8500 \times 3) + (10000 \times 2) + (8000 \times 4)$
Cost = $18000 + 70000 + 1500 + 25500 + 20000 + 32000$
Cost = $\$167,000$

## Question 2: Classifier Calibration

### a. Test error rates

Using the provided training data, LogisticRegression (C=1.0, max_iter=1000) and GaussianNB models were fitted. Their performance on the test set is:
Test error rate (Logistic Regression): 0.1369
Test error rate (GaussianNB): 0.1349

### b. Calibration Curves

The calibration curves for both models were generated using `calibration.calibration_curve` with `n_bins=10`.

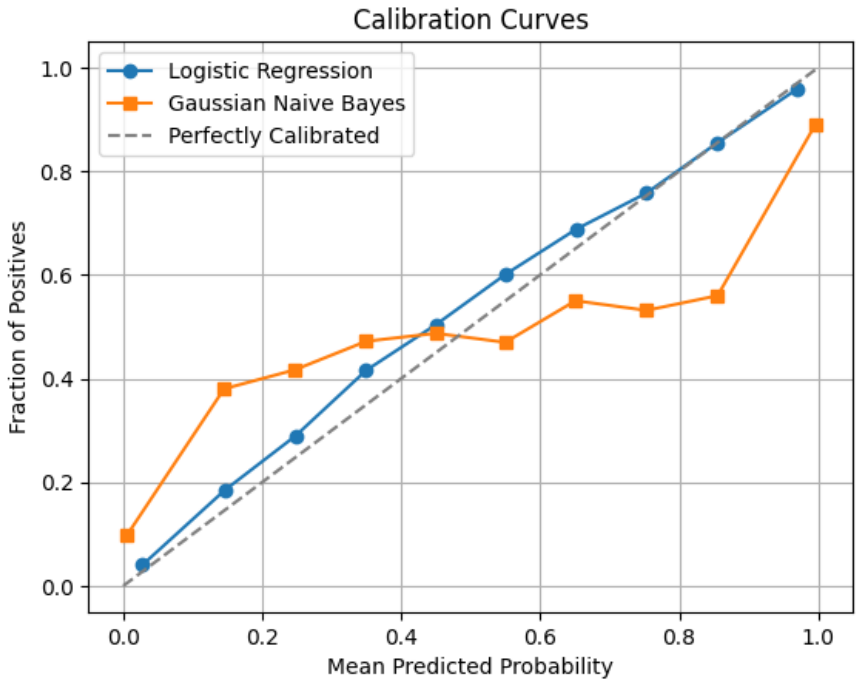


Figure 1: Calibration Curves for Logistic Regression and Gaussian Naive Bayes.

Qualitative Differences: The Logistic Regression curve is much closer to the diagonal line (perfectly calibrated), indicating its predicted probabilities align well with the true frequencies across most bins. It shows slight overconfidence in some mid-range probability bins. The Gaussian Naive Bayes curve deviates significantly from the diagonal. It exhibits overconfidence for predicted probabilities below approximately 0.4 (predicting higher probabilities than observed frequencies) and underconfidence for probabilities above 0.4 (predicting lower probabilities than observed frequencies). The S-shape is characteristic of Naive Bayes models on this type of data.

### c. Expected Calibration Error

The Expected Calibration Error (ECE) was computed using the implemented function:
ECE (Logistic Regression): 0.0201
ECE (Gaussian NB): 0.1063

### d. Reasons for Calibration Differences

Logistic Regression directly models the probability $P(Y = 1|X)$ using the logistic function, which naturally produces well-calibrated probabilities when the model assumptions hold reasonably well and the model is not severely over/underfit. Gaussian Naive Bayes, on the other hand, makes a strong assumption that features are conditionally independent given the class. When this assumption is violated (as it often is in real-world data), the resulting probability estimates calculated via Bayes' theorem can be systematically pushed towards 0 or 1, leading to poor calibration (often appearing overconfident for extreme probabilities and underconfident elsewhere, or vice versa, depending on the data structure).

# Question 3: Imbalanced Classification

## a. Class Proportions

The class proportions for the training and test datasets are visualized below.

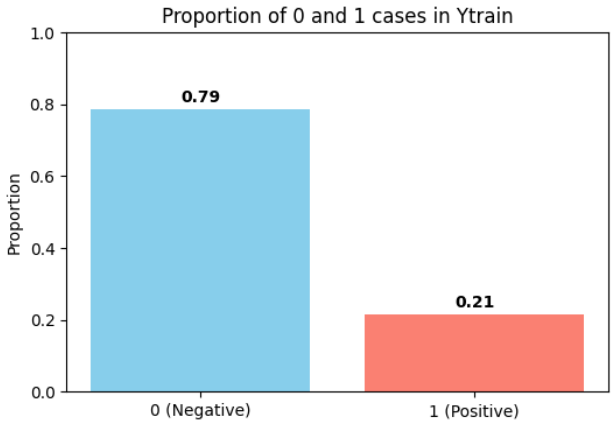


Figure 2: Class Proportions in the Training Data.

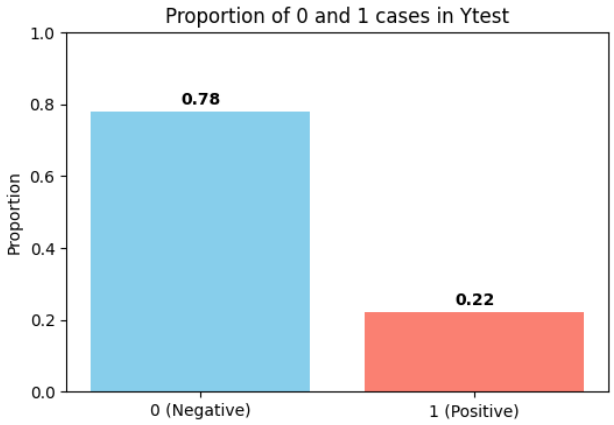


Figure 3: Class Proportions in the Test Data.

Training proportions: red_blood_cell (Class 0): $\approx 0.79$, trophozoite (Class 1): $\approx 0.21$.
Test proportions: red_blood_cell (Class 0): $\approx 0.78$, trophozoite (Class 1): $\approx 0.22$.
Comparison: The test proportions are very similar to the training proportions and do not seem to be significantly different (i.e., not out of distribution). However there is imbalance between the proportion of negative and positive cases.

## b. Base Model Results

The standard MLP model was trained for 50 epochs. The results on the test set are:
Overall error rate: 0.0910
Error rate per class:

red_blood_cell: 0.0364
trophozoite: 0.2832
Balanced Error Rate (BER): 0.1598

### c. Balanced NLL Model Results

The MLP model was updated to use the balanced negative log likelihood loss and trained for 50 epochs. The class weights calculated were approximately $[0.000156, 0.000573]$. The results on the test set are:
Overall error rate: 0.0890
Error rate per class:
red_blood_cell: 0.0729
trophozoite: 0.1460
Balanced Error Rate (BER): 0.1094

### d. Discussion

Yes, learning the model using the balanced negative log likelihood significantly improved the Balanced Error Rate (BER), reducing it from 0.1598 to 0.1094. This improvement came primarily from reducing the error rate on the minority class (trophozoite) from 0.2832 down to 0.1460, at the cost of slightly increasing the error rate on the majority class (red_blood_cell) from 0.0364 to 0.0729. The impact on the overall test error rate was minimal, showing a slight improvement from 0.0910 to 0.0890. Training with balanced NLL successfully encouraged the model to pay more attention to the under-represented class, leading to a more balanced performance across classes.

## Question 4: Experiment Design

### a. Train-Validation-Test Experiment

The learning set was split 80/20 into training and validation sets (random_state=42). Ridge regression was trained for different alphas.
Best alpha: 12.9155
Corresponding Train MSE: 0.0016
Corresponding Validation MSE: 0.5117

### b. 5-Fold Cross-Validation Experiment

5-Fold Cross-Validation (shuffle=True, random_state=42) was used on the learning set to select alpha.
Best alpha: 1.6681
Corresponding Avg. Train MSE: 0.000038 (Note: This is very low, likely due to averaging over folds where the model fits the training part extremely well)
Corresponding Avg. Cross-Validation MSE: 0.4549

### c. Validation MSE vs. Alpha

The validation MSE curves for both experiment designs are shown below.

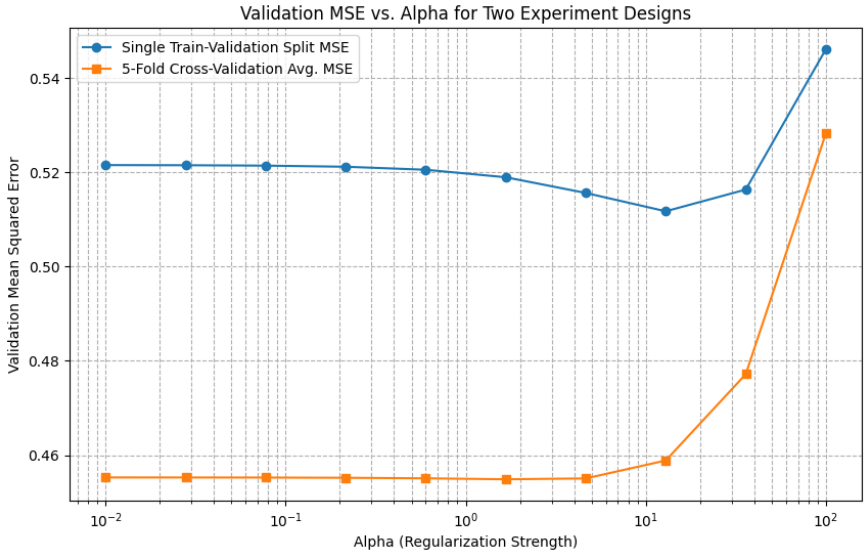


Figure 4: Validation MSE vs. Alpha for Single Split and 5-Fold CV.

### d. Final Model Re-fits and Test MSE

The models were re-fitted on the entire learning set using the best alphas found.
Model 1 (from Train-Val-Test): Best Alpha = 12.9155, Test MSE = 0.3344
Model 2 (from 5-Fold CV): Best Alpha = 1.6681, Test MSE = 0.3101

The 5-fold cross-validation design gave a better (lower) test set MSE (0.3101) compared to the single train-validation-test split (0.3344).

### e. Bonus: Stability Meta-Experiment

Approach Explanation: To assess the stability of the result from part (d), the entire experiment (parts a, b, and d) was repeated 50 times. In each repetition, a different random seed (from 0 to 49) was used for both the initial 80/20 train-validation split (for method 4a) and the KFold shuffling (for method 4b). The best alpha was determined independently by each method within each repetition, the models were retrained on the full learning set with their respective best alphas, and the final test MSE was recorded for both. This allows us to see how often each hyperparameter selection strategy leads to a better final test model under different random partitions of the learning data.

Meta-Experiment Results (50 trials):
5-Fold CV produced a better Test MSE in 21/50 trials (42.0%).
Single Train-Val-Test produced a better Test MSE in 29/50 trials (58.0%).
Average Test MSE (CV): 0.3206 (Std: 0.0132)
Average Test MSE (TVT): 0.3312 (Std: 0.0295)

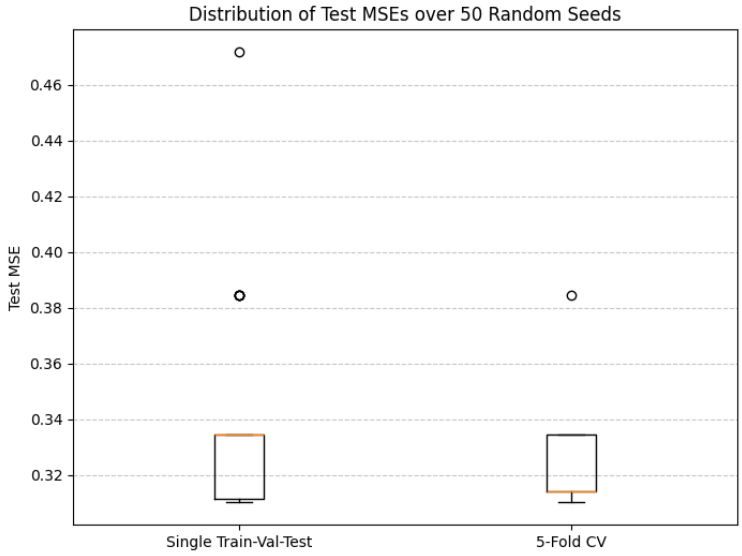


Figure 5: Distribution of Test MSEs over 50 Random Seeds.

Discussion: Although the single train-val-test split yielded a better test MSE more often (58% of trials) in this meta-experiment, the average test MSE for the 5-Fold CV method was lower (0.3206 vs 0.3312). Furthermore, the standard deviation of the test MSEs for the CV method was much smaller (0.0132) compared to the single split method (0.0295). This suggests that while a single split might occasionally get lucky and find a better hyperparameter due to the specific validation set chosen, the 5-Fold CV method is more stable and reliable, producing models with lower error on average and less variability depending on the random partitioning.

## Question 5: Generative AI Use

I used AI tools to assist with coding certain parts, specifically for method syntax and plotting with Matplotlib.