Risk Modeling and Classification

EN5422/EV4238 | Fall 2023

w05_classification_2.pdf

(Week 5 – 2/2)

Contents

1 Evaluating Binary Risk Models

1.1 Common Binary Loss Function

- Suppose we are going to predict a binary outcome $Y \in \{0,1\}$ with $\hat{p}(x) \in \{0,1\}$.
    - Call $\hat{p}(x)$ *the risk score*
- **Brier Score / Squared Error**

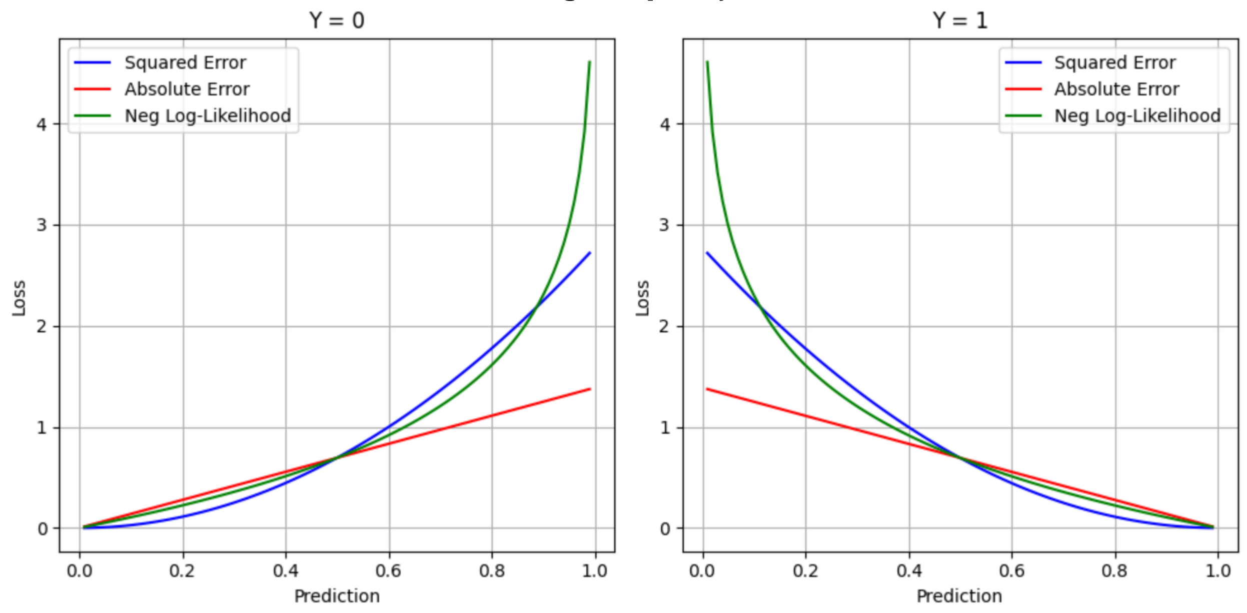
$$L(y,\hat{p}) = (y-\hat{p})^2 = \begin{cases} (1-\hat{p})^2 & y=1 \\ \hat{p}^2 & y=0 \end{cases}$$

- **Absolute Error**

$$L(y,\hat{p}) = |y-\hat{p}| = \begin{cases} 1-\hat{p} & y=1 \\ \hat{p} & y=0 \end{cases}$$

- **Bernoulli negative log-likelihood (Log-Loss)**
    - This is the loss function for Logistic Regression

$$\begin{aligned} L(y,\hat{p}) &= -\{y\log\hat{p} + (1-y)\log(1-\hat{p})\} \\ &= \begin{cases} -\log\hat{p} & y=1 \\ -\log(1-\hat{p}) & y=0 \end{cases} \end{aligned}$$



Note: Log-Loss heavily penalizes predictions that are confident but wrong. For instance, predicting a probability of 0.99 for an event that does not happen would result in a high log loss.

1.2 Model Comparison

Code Example

```
# Evaluation function
def evaluate(p_hat, y):
    return {
        'mn_log_loss': -log_loss(y, p_hat),
        'mse': mean_squared_error(y, p_hat),
        'mae': mean_absolute_error(y, p_hat)
    }

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=2000,
random_state=321)

# Fit logistic regression on training data
lm = LogisticRegression(max_iter=5000)
lm.fit(X_train, y_train)
p_hat_lm = lm.predict_proba(X_test)[:, 1]

print(evaluate(p_hat_lm, y_test))

# Fit lasso logistic regression with cross-validation
lasso = LogisticRegressionCV(penalty='l1', solver='liblinear', cv=10,
max_iter=5000)
lasso.fit(X_train, y_train)
p_hat_lasso = lasso.predict_proba(X_test)[:, 1]

print(evaluate(p_hat_lasso, y_test))
```

```
{'mn_log_loss': -0.08208688510160332, 'mse': 0.020984697300701974, 'mae': 0.04180403130611189
{'mn_log_loss': -0.08138061351520372, 'mse': 0.020842536246359983, 'mae': 0.04292240657380746
```

1.3 Calibration

A risk model is said to be *calibrated* if the predicted probabilities are equal to the true risk (probabilities).

$$\Pr(Y = 1 \mid \hat{p} = p) = p \qquad \text{for all } p$$

Note

Calibration in the context of a risk model refers to the idea that the predicted probabilities from a model should match the actual outcomes in the real world, especially over a large number of predictions.

Example:

Imagine you have a weather forecasting model that predicts the probability of rain tomorrow.

1. On 100 days, your model predicted a 70% chance of rain the next day.
2. If the model is well-calibrated, it would mean that out of those 100 days, it actually rained on approximately 70 of them.

So, the model's predicted probability (70% chance of rain) should be close to the true frequency of the event happening (rain on 70 out of 100 days).

In summary, a calibrated model provides trustworthy probabilities. If it says there is an X% chance of an event happening, then over many occurrences, that event should happen approximately X% of the time.

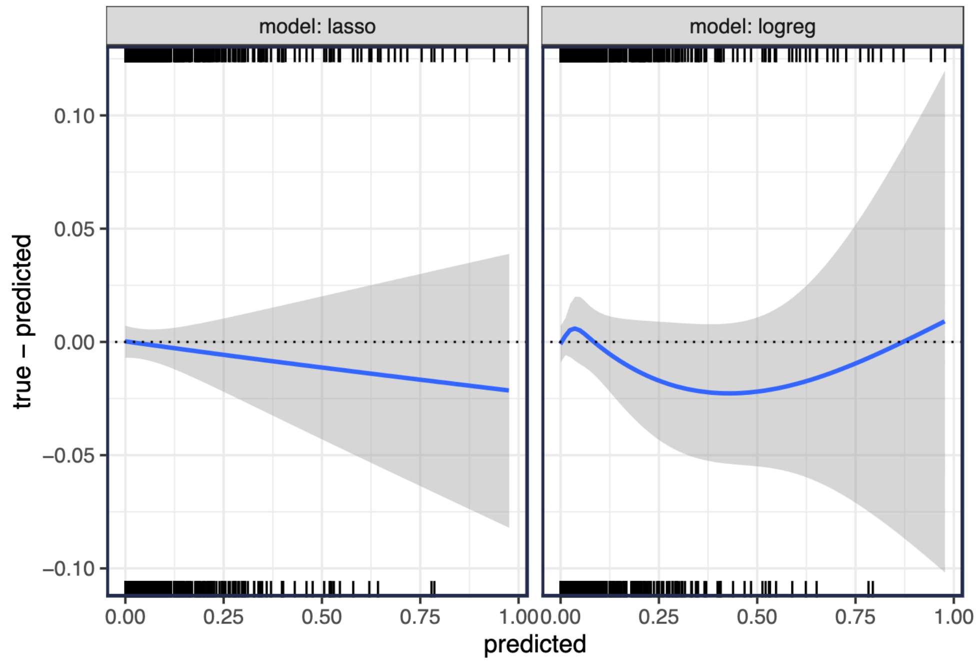
$\Pr(Y = 1 \mid \hat{p} = p) = p$ is the probability that the outcome Y is 1 (or the event happens) given that the predicted probability is $\hat{p}$. p is a specific probability value, such as 0.70 for 70%.

Interpretation: For a model to be calibrated, the conditional probability of the event happening (e.g., it raining) given a predicted probability should be equal to that predicted probability, across all possible predicted probabilities.

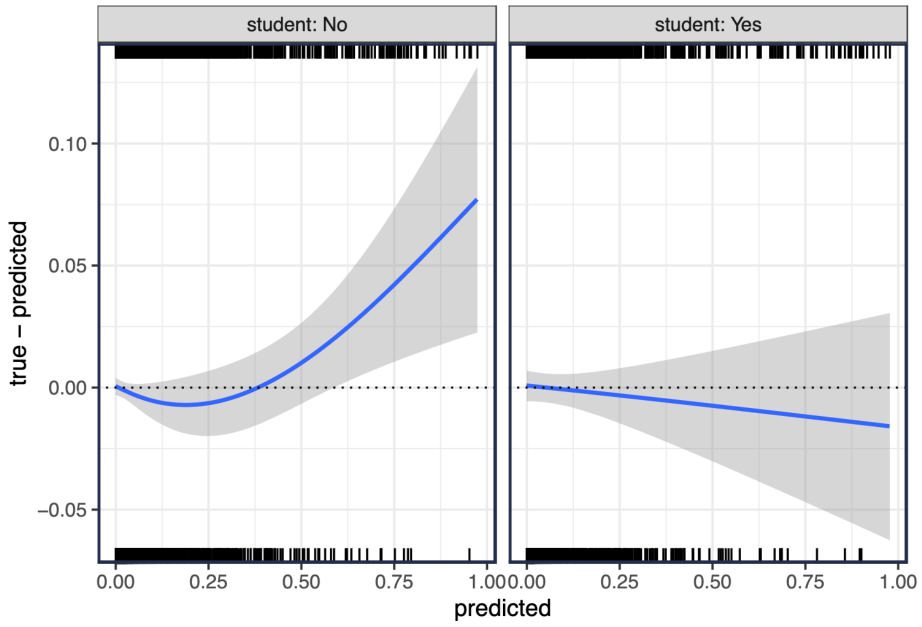
Using our earlier example:

If our weather model predicts a 70% chance of rain tomorrow ($\hat{p} = 0.70$), then the true probability of it raining tomorrow, based on many such predictions, should also be 70% ($\Pr(Y = 1 \mid \hat{p} = 0.7) = 0.70$).

In other words, whenever the model says there is a "*p*" chance of something happening, the actual chance of that event occurring should be "*p*". If this holds true for all values of "*p*", then the model is calibrated.



Calibration plots can be used to measure drift, fairness, and model/algorithmic bias. Consider comparing the predictive performance of our models for Students and Non-Students.



1.3.1 Estimating Calibration

To measure mis-calibration, we can treat the predictions as features and use the predictions as an offset. E.g., to check for linear deviation.

$$\operatorname{logit} p(x) = \beta_0 + \beta_1 \hat{p}(x) + \operatorname{logit} \hat{p}(x)$$

fit on a hold-out set, and check how far β_0 and β_1 are from 0.

We will revisit calibration when we cover Support Vector Machines (SVM) to convert a generic score to a probability.
Some additional resources:

- Platt Scaling
- Isotonic Regression
- Calibration: the Achilles heel of predictive analytics
  - Article on clinical decision-making
- The Measure and Mismeasure of Fairness: A Critical Review of Fair Machine Learning

1.4 Area Under the Receiver Operating Characteristic Curve (AUC or AUROC or C-Statistic)

The AUROC of a risk model is: the probability that the model will rank a randomly chosen positive example ($Y = 1$) higher than a randomly chosen negative example ($Y = 0$) i.e.,

$$\text{AUROC} = \Pr\big(\hat{p}(x_1) > \hat{p}(x_0)\big)$$

- where x_k is a randomly chosen example from class $Y = k$.

To estimate the AUROC you will fit a model to training data and make predictions on hold-out (test) data with known labels. Hopefully the model will assign large probabilities to the outcome of interest ($Y = 1$) and low probabilities to the other class. Then compare the probabilities between all pairs of observations where one comes from the $Y = 1$ set and the other from the $Y = 0$ set. The AUROC is the proportion of the pairs where the estimated probability for the outcome of interest is larger than the probability for the other outcome. The extra term is to handle ties in predicted probability.

$$\widehat{\text{AUROC}} = \frac{1}{n_1 n_0} \sum_{i:y_i=1} \sum_{j:y_j=0} \left(\mathbb{I}(\hat{p}_i > \hat{p}_j) + \frac{1}{2}\mathbb{I}(\hat{p}_i = \hat{p}_j) \right)$$

The AUROC assesses the discrimination ability of the model. It gives a different assessment on model performance from calibration. Notice that the AUROC is the same for any monotonic transformation of the estimated probabilities. E.g., we can use $\hat{p}$ or $\log \hat{p}$ or $\text{logit}(\hat{p})$ or $\frac{\hat{p}}{10}$ and still get the same AUROC. But calibration assesses how closely the estimated probabilities match the actual probabilities as well s helping to identify the regions in feature space where the predictions are poor.

- A helpful discussion on AUROC: https://stats.stackexchange.com/questions/145566/how-to-calculate-area-under-the-curve-auc-or-the-c-statistic-by-hand
- We will say more about ROC curves later in the notes.

Code Example
1. We will first generate some sample data. 2. Then, split the data into training and test sets. 3. Fit a model (in this case, a logistic regression) to the training data. 4. Make predictions on the test data. 5. Compute the AUROC using the provided formula.

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Generate some sample data
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2,
random_state=321)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=321)

# Fit a logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict probabilities on the test set
y_pred_proba = model.predict_proba(X_test)[:, 1]  # Getting the probability
estimates of class 1

# Compute the AUROC using the given formula
n_1 = sum(y_test)
n_0 = len(y_test) - n_1
count = 0

for i in range(len(y_test)):
    for j in range(len(y_test)):
        if y_test[i] == 1 and y_test[j] == 0:
            if y_pred_proba[i] > y_pred_proba[j]:
                count += 1
            elif y_pred_proba[i] == y_pred_proba[j]:
                count += 0.5

auroc_hat = count / (n_1 * n_0)
print(f"Estimated AUROC: {auroc_hat}")

# Note: In practice, we often use sklearn's roc_auc_score function to compute AUROC,
# but for the purpose of this explanation, we're computing it manually.
Estimated AUROC: 0.9141713014460512
```

This script will:

1. Generate a dataset with 1,000 samples and 20 features.
2. Split the dataset into training and test sets.
3. Train a logistic regression model on the training data.
4. Predict the probabilities of class `1` on the test data.
5. Compute the AUROC using the provided formula.

```
# Compute the AUROC using sklearn's roc_auc_score
from sklearn.metrics import roc_auc_score
auroc = roc_auc_score(y_test, y_pred_proba)
print(f"AUROC: {auroc}")
AUROC: 0.9141713014460511
```

2 Risk Scoring vs. Classification

Most of the models we will encounter can output a predicted probability $\hat{p}_i(x) = \widehat{\Pr}(Y = k|X = x)$ for every class $k \in \mathcal{G}$.

Sometimes a *hard classification* needs to be made i.e., decide on single label/class to assign the observation.

1. Hard Classification:
   - Use training data to estimate the *label* $\hat{G}(X)$
   - The *loss/cost* $L\left(G, \hat{G}(X)\right)$ is the loss incurred by estimating G with $\hat{G}$.
2. Risk Scoring (Soft-Classification):
   - Use training data to estimate the *probability* $\hat{p}_i(X)$.
   - The loss/cost $L\left(G, \hat{G}(X)\right)$ is the loss incurred by estimating G with $\hat{p}_i(X)$, where $\hat{p}(X) = [\hat{p}_1(X), \dots, \hat{p}_k(X)]$.
3. Ranking:
   - Use the training data to rank the test observations according to estimated risk level.
   - The loss/cost is based on the number of outcomes of interest in the top proportion of risk.

2.1 Example:

Predicting Disease Status: Imagine a medical dataset where we want to predict whether a patient has a specific disease or not based on several features (like age, symptoms, medical history, etc.). The outcome can be 'Disease' or 'No Disease'.

Code Example

1. Hard Classification:

In this approach, the model will directly predict the label for each patient without giving any probabilities.

Model Output:

- Patient A: Disease
- Patient B: No Disease
- Patient C: Disease

Loss/Cost: Suppose Patient A truly had the disease, Patient B did not, and Patient C also did not. In this case, the model's prediction for Patient C was wrong. The loss would be the penalty for this incorrect classification.

2. Risk Scoring (Soft-Classification):

Instead of directly predicting the disease status, the model outputs the probability that each patient has the disease.

Model Output:

- Patient A: 0.8 (meaning 80% probability of having the disease)
- Patient B: 0.1 (10% probability of having the disease)
- Patient C: 0.65 (65% probability)

Loss/Cost: Here, the loss could be calculated using something like Log-Loss. It penalizes not just incorrect classifications but also how "off" the model's probabilities are from the true outcomes.

3. Ranking:

Instead of outputting direct classifications or probabilities, the model may rank the patients based on their risk levels. This approach is particularly useful in scenarios like prioritizing patients for further medical tests based on their risk scores.

Model Output:

1. Patient A (highest estimated risk)
2. Patient C
3. Patient B (lowest estimated risk)

Loss/Cost: Imagine if in reality, the top 2 patients (based on some ground truth risk or outcome) were Patient A and Patient B. The model's ranking made an error by placing Patient C above Patient B. The loss could be calculated based on such ranking errors.

```python
from sklearn.linear_model import LogisticRegression

# Toy data
X = [[50, 1], [30, 0], [40, 1]]  # [age, symptom_present]
y = [1, 0, 1]  # 1: Disease, 0: No Disease

# Train a logistic regression classifier
clf = LogisticRegression()
clf.fit(X, y)

# Predict classes
predictions = clf.predict(X)
```

```
print(predictions)  # e.g., [1, 0, 1]



# Using the same toy data and trained classifier from above

# Predict probabilities
probabilities = clf.predict_proba(X)

print(probabilities)  # e.g., [[0.2, 0.8], [0.9, 0.1], [0.35, 0.65]]



# Using the probabilities from the soft-classification step

# Extracting probabilities of having a disease
disease_probabilities = [prob[1] for prob in probabilities]

# Rank patients by risk
ranking = sorted(range(len(disease_probabilities)), key=lambda k:
disease_probabilities[k], reverse=True)

print(ranking)  # e.g., [0, 2, 1], indicating Patient A is highest risk,
followed by Patient C, then Patient B
```

3 Binary Classification

3.1 Decision Theory

- We are considering *binary* outcome, so the outcomes $G \in \{0,1\}$
- Let $\hat{p} = \Pr(G = 1|X = x)$
- Loss Function: $L(\text{True Label}, \text{Estimated Label}) = L(G, \hat{G})$
- A model's *Expected Prediction Error (EPE)* (also called *Risk)* at input *X* is the expected loss on new data with input *X*.
- The EPE (for a binary outcome) is:

$$\begin{aligned}\text{EPE}_x(g) &= E_{(G|X=x)}\big[L\big(G, \hat{G}(x) = g\big)\,\big|\,X = x\big]\\ &= L(1,g)\Pr(G = 1\,|X = x) + L(0,g)(1 - \Pr(G = 1\,|\,X = x))\\ &= L(1,g)p(x) + L(0,g)\big(1 - p(x)\big)\end{aligned}$$

Code Example

```
import numpy as np
from sklearn.linear_model import LogisticRegression

# Toy data
X = [[50, 1], [30, 0], [40, 1]]  # [age, symptom_present]
```

```
y = [1, 0, 1]  # 1: Disease, 0: No Disease

# Train a logistic regression classifier
clf = LogisticRegression()
clf.fit(X, y)

# For demonstration, we'll compute EPE for the instance [40, 1] and prediction g = 1
x_instance = [[40, 1]]
g = 1

# Calculate p(x), the estimated probability that G=1 given X=x
p_x = clf.predict_proba(x_instance)[0][1]

# Define the log-loss function L
def log_loss(y_true, y_pred):
    # Clip predictions to avoid log(0)
    epsilon = 1e-15
    y_pred = np.clip(y_pred, epsilon, 1 - epsilon)

    return - (y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))

# Compute EPE
EPE_x = log_loss(1, g) * p_x + log_loss(0, g) * (1 - p_x)

print(f"EPE for the instance {x_instance} with prediction g={g} is: {EPE_x}")
```

- Hard Decision ($\hat{G}(x) \in \{0,1\}$): chose $\hat{G}(x) = 1$ if

$$\text{EPE}_x(1) < \text{EPE}_x(0)$$

$$L(1,g)p(x) + L(0,1)\big(1 - p(x)\big) < L(1,0)p(x) + L(0,0)\big(1 - p(x)\big)$$

$$p(x)\big(L(1,1) - L(1,0)\big) < \big(1 - p(x)\big)\big(L(0,0) - L(0,1)\big)$$

$$p(x)\big(L(1,0) - L(1,1)\big) \geq \big(1 - p(x)\big)\big(L(0,1) - L(0,0)\big) \qquad (multiply\ both\ sides\ by - 1)$$

$$\frac{p(x)}{1 - p(x)} \geq \frac{L(0,1) - L(0,0)}{L(1,0) - L(1,1)}$$

Note: In most cases, there will be no loss/cost for making a correct classification. Thus, it is convention to set $L(0,0) = L(1,1) = 0$ in these scenarios.

Code Example

```
def hard_decision(p_x, L):
    """
    Make a hard decision based on the predicted probability and loss values.

    Parameters:
    - p_x: Predicted probability for class 1.
    - L: Loss matrix.
        L[0][0] is the loss for predicting 0 when true value is 0.
        L[0][1] is the loss for predicting 1 when true value is 0.
        L[1][0] is the loss for predicting 0 when true value is 1.
        L[1][1] is the loss for predicting 1 when true value is 1.

    Returns:
    - 1 if the decision is to classify as class 1, 0 otherwise.
    """
    left_side = p_x / (1 - p_x)
    right_side = (L[0][1] - L[0][0]) / (L[1][0] - L[1][1])

    return 1 if left_side >= right_side else 0

# Example usage
L = [[0, 1], [1, 0]]  # Typical 0-1 loss matrix
p_x = 0.7  # Sample predicted probability for class 1

decision = hard_decision(p_x, L)
print(f"Hard Decision for p(x) = {p_x}: G_hat(x) = {decision}")
```

3.1.1 Example: Cancer Diagnosis

- Say, we have a goal of estimating if a patient has cancer using medical imaging.
    - Let $G = 1$ for cancer and $G = 0$ for no cancer.
- Suppose we have solicited a loss function with the following values:
    - $L(G = 0, \hat{G} = 0) = 0$: There is no loss for correctly diagnosis a patient without cancer.
    - $L(G = 1, \hat{G} = 1) = 0$: There is no loss for correctly diagnosis a patient with cancer.
    - $L(G = 0, \hat{G} = 1) = C_{FP}$: There is a cost of C_{FP} units if the model issues a *false positive,* estimating the patient has cancer when they don't.
    - $L(G = 0, \hat{G} = 1) = C_{FN}$: There is a cost of C_{FN} units if the model issues a *false negative*, estimating the patient does not have cancer when they really do.
    - In these scenario C_{FN} is often much larger than C_{FP} ($C_{FN} \gg C_{FP}$) because the effects of not promptly treating (or further testing, etc.) a patient is more severe than starting a treatment path for patients that don't actually have cancer.

- The optimal decision is to issue a positive indication for cancer if $\text{EPE}_x(1) < \text{EPE}_x(0)$. This occurs when:

$$\frac{p(x)}{1-p(x)} \geq \frac{C_{FP}}{C_{FN}} \quad \text{OR} \quad p(x) \geq \frac{C_{FP}}{C_{FP}+C_{FN}} \quad \text{OR} \quad \log\left(\frac{p(x)}{1-p(x)}\right) \geq \log(\frac{C_{FP}}{C_{FN}})$$

- The ratio of C_{FP} to C_{FN} is all that matters for the decision. Let's say that $C_{FP} = 1$ and $C_{FN} = 10$. Then if $p(x) \geq 1/11$, our model will diagnose cancer.
  - Note: $p(x) = \Pr(Y = 1 \,|X = x)$ is affected by the class prior $\Pr(Y = 1)$ (e.g., the portion of the population tested who have cancer), which is usually going to be small.

3.1.2 Optimal Threshold

- Recall, the optimal *hard classification* decision is to choose $\hat{G} = 1$ if:

$$\frac{p(x)}{1-p(x)} \geq \frac{L(0,1)-L(0,0)}{L(1,0)-L(1,1)}$$

- It can be convenient to use model output other than $\frac{p(x)}{1-p(x)}$ to make decisions.
- Some model directly output $\hat{p}(x)$
- Other models, like GLMs, naturally work with the link function (linear part)
  - Denote $\gamma(x)$ as the *logit* of $p(x)$
- Table of equivalent representations:

Score	Threshold	Threshold (simplified)
$\frac{p(x)}{1-p(x)}$	$\frac{L(0,1)-L(0,0)}{L(1,0)-L(1,1)}$	$\frac{C_{FP}}{C_{FN}}$
$\gamma(x) = \log\frac{p(x)}{1-p(x)}$	$\log\frac{L(0,1)-L(0,0)}{L(1,0)-L(1,1)}$	$\log\left(\frac{C_{FP}}{C_{FN}}\right)$
$p(x)$	$\log\frac{L(0,1)-L(0,0)}{L(1,0)-L(0,0)-L(1,0)-L(1,1)}$	$\frac{C_{FP}}{C_{FP}+C_{FN}}$

The Threshold (simplified) assumes $L(0,0) = L(1,1) = 0$

3.1.3 Using estimated values

- We will never have the actual $p(x)$ or $\gamma(x)$, so replace them with the estimated values.
- For a given threshold t, and input x, the hard classification rule is $\hat{G}_t = \mathbb{I}(\hat{\gamma}(x) \geq t)$

Note
Because we have to estimate $\hat{p}(x)$ or $\hat{\gamma}(x)$, the best threshold t* may differ from the theoretical optimal and need to be estimated. (more information about this later).