CPE 490 590: Machine Learning for Engineering Applications

08 Logistics Regression

Rahul Bhadani

Electrical & Computer Engineering, The University of Alabama in Huntsville

Outline

Announcement

Quiz

- Quiz 2: Opens on Feb 21, 2025 Midnight, Closes on Feb 23, 2025 11:59 PM
- Covers Topics on Linear Regression
- Duration: Approximately One Hour
- Requires you running Python Notebook while taking the quiz for some questions.

First Thing First

Logistics Regression is not a *Regression* but a **classification** method.

Motivation

Motivating Logistic Regression

- Linear regression provides us a measure of how close a sample is to the decision boundary from the function $g(\mathbf{x})$. That is the larger $|g(\mathbf{x})|$ the farther a sample is from the plane.
- Unfortunately, this score does not directly correspond to the probability that a sample belongs to a class.
  - We could try to think of a heuristic to connect $g(\mathbf{x})$ to a probability; however, let us try to derive a relationship.
  - We need to revisit Bayes to get the conversation of logistic regression started.
- Linear regression ended up having a very convenient solution (i.e., it was a convex optimization task) and we want to try to obtain a similar results – *if possible.*

Goal

Start with Bayes theorem to find a linear predictive model that can provide an estimate of the posterior probability.

Reivisiting Bayes' Theorem

The Sum Rule in Probability

The marginal probability of a single random variable can be obtained by summing (integrating) the probability density function (pdf) over all values of all other variables. For example,

$$P(X) = \sum_{Y \in \mathcal{Y}} P(X, Y)$$

$$P(X) = \sum_{Z \in \mathcal{Z}} \sum_{Y \in \mathcal{Y}} P(X, Y, Z)$$

The Product Rule in Probability

The joint probability can always be obtained by multiplying the conditional probability (conditioned on one of the variables) with the marginal probability of the conditioned variable

$$P(X, Y) = P(Y)P(X|Y) = P(X)P(Y|X)$$

Bayes' Rule

The combination of these two rules gives us Bayes Rule. Let Y denote the outcome that we seek to predict (e.g., healthy or not) and X denote the variable(s) we are provided (e.g., medical records)

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} = \frac{P(X|Y)P(Y)}{\sum_{y\in\mathcal{Y}} P(X, Y=y)} = \frac{P(X|Y)P(Y)}{\sum_{y\in\mathcal{Y}} P(X|Y=y)P(Y=y)}$$

Some Terminology in Bayes' Rule

In the context of

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

- $P(Y)$ is called **prior** distribution.
- $P(X|Y)$ is called the **likelihood**.
- $P(Y|X)$ is the posterior distribution.
- $P(X)$ is treated as constant for the known data and is marginal probability distribution. It can be treated as a normalizing constant.

Bayesian Prior to Posterior

We could rewrite as

$$P(parameters|data) = \frac{P(data|parameters)P(parameters)}{P(data)}$$

$$posterior \propto normalized\ \ likelihood \times prior$$

How to Get Prior

- Past data: ”today’s posterior = tomorrow’s prior”
- Meta-analysis of past data: amalgamation of multiple sources
- Expert knowledge
- Individual subjective opinion

Example: Bayes Rule for Email Classification

Consider $X \in \{0, 1\}^W$, a vector of unique words, their value being "0" or "1" if the word appears in an email or not. This is known as a "bag of words" model.

Example: Bayes Rule for Email Classification

We can write:

$$\begin{aligned} & P(Y = \text{awesome} | Nigerian = 1, Prince = 1, Love = 0, \ldots) \\ = & \frac{P(Y = \text{awesome}) P(Nigerian = 1, Prince = 1, Love = 0, \ldots | y)}{P(Nigerian = 1, Prince = 1, Love = 0, \ldots)} \end{aligned}$$

Given our prior knowledge about the words “Nigerian” and “Prince” occurring in spam emails, it will be unlikely that this is an “awesome” email.

Making Decision with Bayes

Observation: X.

$\omega \in \Omega$: true class label.

ω^*: class labels that we may predict using Bayes when we have the largest posterior probability.

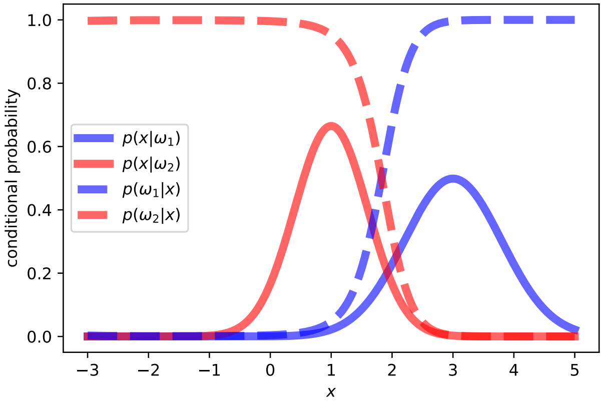
Selecting a Class using Bayes'

Formally, we have

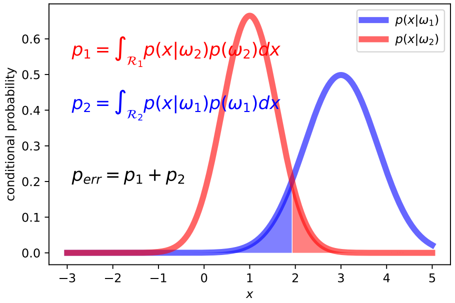
$$\omega^* = \arg\max_{\omega\in\Omega} P(\omega|X) = \arg\max_{\omega\in\Omega} \frac{P(X|\omega)P(\omega)}{P(X)} = \arg\max_{\omega\in\Omega} P(X|\omega)P(\omega)$$

where $P(X)$ can be omitted since it only acts as a normalization constant.
arg max is the argument of maxima which means the value ω at which $P(X|\omega)P(\omega)$ is the maximum.

Visualizing the Posterior and Likelihood



Probability of Error



Bayes has the minimum probability of error for any classifier of a random outcome. We write probability of error as

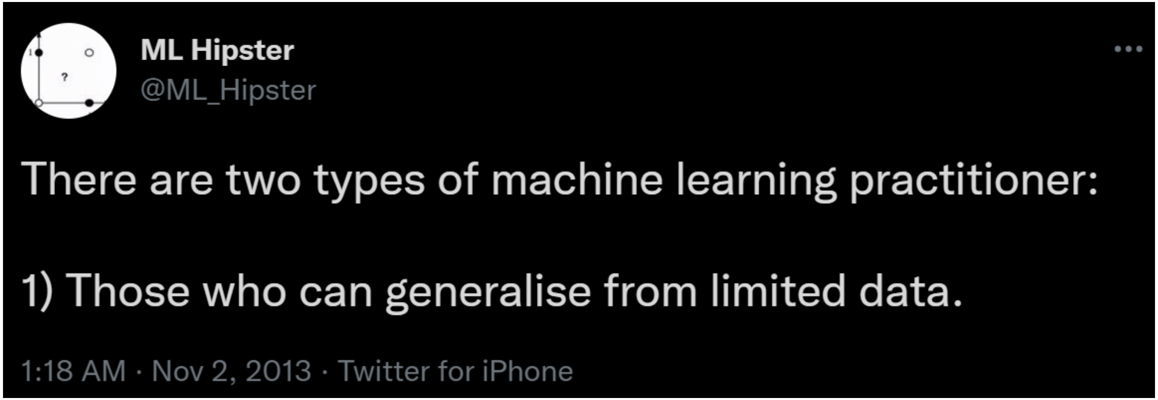
$$P(\text{error}|\mathbf{x}) = 1 - P(\omega^*|\mathbf{x})$$

Decision Making with Bayes

- The Bayes decision rule (i.e., choose the outcome with the largest posterior) leads to the smallest probability of error.
- Sometimes it is easier to work with the log of the posterior for numerical reasons and it leads to the same outcome since the log is a monotonically increasing function
- **Question**: If the Bayes decision rule has the smallest probability of error, why do we need machine learning?

Why do we need ML if we have Bayes?

- It is true that Bayes rule leads to the smallest probability of error; however, for this to be the case then we need to have access to $P(Y)$ and $P(X|Y)$. Or we need a very good estimate.
  - In reality, estimating these quantities can be very challenging – or impossible – with (1) making some assumptions or (2) a LOT of data.
  - Example: Assume the data are Gaussian then estimate μ and Σ? How much data do we need to estimate these parameters? What if the data are not Gaussian?
- In practice, we need to make assumptions and learn to deal with limited data.



ML Hipster
@ML_Hipster

There are two types of machine learning practitioner:

1) Those who can generalise from limited data.

1:18 AM · Nov 2, 2013 · Twitter for iPhone

Naïve Bayes

The Naïve Bayes Approach

Bayes rules states that we choose the class based on $p(\mathbf{x}|Y)P(Y)$, but estimating $p(\mathbf{x}|Y)P(Y)$ is *very difficult*. So assume the features are conditionally independent!

$$P(Y|\mathbf{x}) \propto p(\mathbf{x}|Y)P(Y) = p(x_1, \dots, x_D|Y)P(Y) = P(Y)\prod_{j=1}^{D} p(x_j|Y)$$

- The advantage is that we need to estimate D 1-dimensional distributions, not the D dimensional joint distribution.
- Is this assumption correct? Probably not, but it works well in many applications.

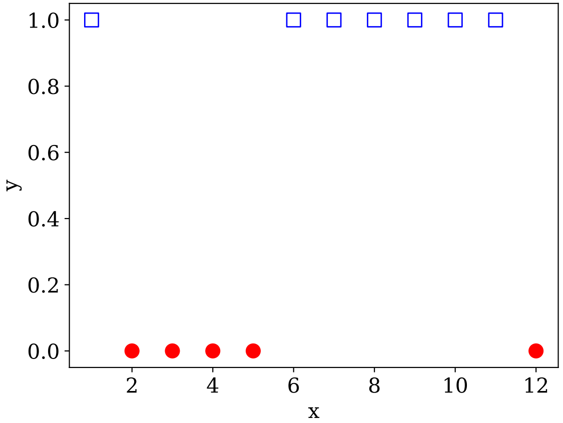
Motivating Logistic Regression

A Dataset with Two Classes

Consider the dataset given in the table:

x	1	2	3	4	5	6	7	8	9	10	11
y	1	0	0	0	0	1	1	1	1	1	1

A scatter plot for this dataset

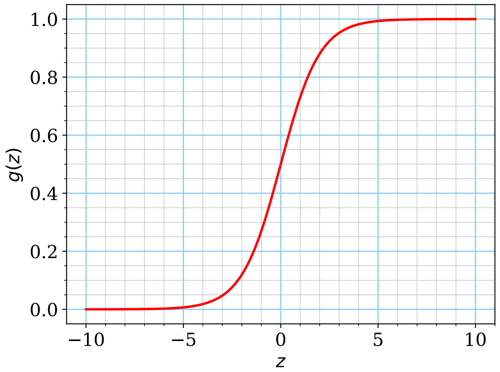


Clearly, distinguishing red and blue points won't work if we use the Linear Regression method. Also, linear regression predicts a continuous value, hence it won't be able to predict either 0 or 1.

Sigmoid Function

By looking at the data, we think we can classify or correctly label the data points if we have a function that is low (0) for some low values of x, and high (1) for some values of x. One such function is **Sigmoid Function** or **Logistics Function** ($g(z)$ in the graph).

$$g(z) = \frac{1}{1 + e^{-z}}$$



Sigmoid Function

We see that

$$\lim_{z \to -\infty} g(z) = 0$$

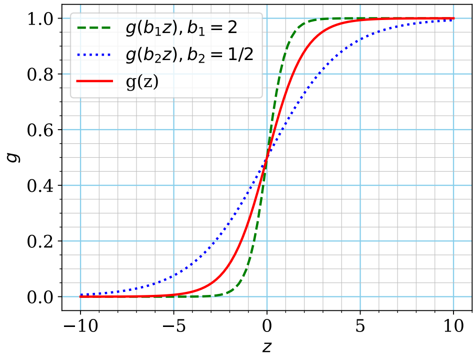
and

$$\lim_{z \to \infty} g(z) = 1$$

It means, that for a very high positive value of x, the function approaches 1, and for a very high negative value of x, the function approaches 0.

Generalized Sigmoid Function

Now, what if we want the slope to change faster or slower? How should we modify the Sigmoid function?



$$g(z) = \frac{1}{1 + e^{-z \cdot b}}$$

Defining Sigmoid Function for Logistics Regression

In the above example b_1, b_2 and $\pm a$ are learnable parameters that we are going to learn from data the same way we learned w_1 and w_0 in the linear regression. Thus, we consider a parameter vector $\boldsymbol{\theta}$ that consists of a vector of parameters to be learned using data $\mathbf{x}$ that we can use to write a generalized version of sigmoid that is data-dependent by setting $z = \boldsymbol{\theta}^\top \mathbf{x}$

$$h_\theta(\mathbf{x}) = g(\boldsymbol{\theta}^\top \mathbf{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^\top \mathbf{x}}}$$

Logistics Regression

Problem Statement in Logistics Regression

The overall goal in Logistics Regression becomes finding $P(Y|X)$, i.e, given information about the random variable X that gives the distribution of the predictor variable, find the probability that the random variable Y takes a certain value.

Assumptions in Logistics Regressions

- **Linearity:** The model assumes that the relationship between the natural log of the probabilities (of each outcome and your predictor variable is linear.
- **No Outliers:** Logistics regression is very sensitive to outliers.
- **Independence:** Each observation point should be independent of each other.

Logistic Regression

From the earlier formulation, h_θ estimates the probability that the data points belong $Y = 1$, and hence we can write it as

$$P(Y = 1 | X = x) = h_\theta(\mathbf{x})$$

$$P(Y = 0 | X = x) = 1 - h_\theta(\mathbf{x})$$

Cost Function

Just like linear regression, we need a cost function based on which we want to estimate $\boldsymbol{\theta}$. Here, we have two scenarios:

1. $y_i = 1$:
   - if $h_{\theta}(\mathbf{x}) \approx 1$, cost ≈ 0 (no penalty)
   - if $h_{\theta}(\mathbf{x}) \approx 0$, cost $\approx$ very high
2. $y_i = 0$:
   - if $h_{\theta}(\mathbf{x}) \approx 1$, cost $\approx$ very high
   - if $h_{\theta}(\mathbf{x}) \approx 0$, cost ≈ 0 (no penalty)

Then as the outcome is 0 or 1, it follows the Bernoulli Distribution.

Bernoulli Distribution Revisited

The Bernoulli distribution is a special case of the binomial distribution where a single trial is conducted (so *n* would be 1 for such a binomial distribution).

The formula for pmf associated with a Bernoulli random variable over possible outcomes *x* is given as follows:

$$f(x;p) = \begin{cases} p & \text{if } x = 1 \\ 1-p & \text{if } x = 0 \end{cases}$$

We can also express this formula as,

$$f(x,p) = p^x(1-p)^{1-x}, \quad x \in \{0,1\}$$

Likelihood Function

Hence, probability of assigning a label y based on the data $\mathbf{x}$ is given by

$$P(Y = y|X = \mathbf{x}) = h_\theta(\mathbf{x})^y \Big[1 - h_\theta(\mathbf{x})\Big]^{(1-y)}$$

Now, we write taking all data into account (all data points are independent):

$$L(\theta) = \Pi_{i=1}^n P(Y = y_i|X = \mathbf{x}_i) = \Pi_{i=1}^n h_\theta(\mathbf{x}_i)^{y_i} \Big[1 - h_\theta(\mathbf{x}_i)\Big]^{(1-y_i)}$$

This is called **likelihood function**.

Log-Likelihood Function

Now, take the log of the likelihood function, which is called the log-likelihood function:

$$LL(\boldsymbol{\theta}) = \sum_{i=1}^{n} y_i \log(h_{\boldsymbol{\theta}}(\mathbf{x}_i)) + (1 - y_i) \log \left[1 - h_{\boldsymbol{\theta}}(\mathbf{x}_i)\right]$$

This is also called **cross-entropy loss**.

Optimization Function

In the logistics regression, our goal becomes maximizing the log-likelihood by choosing the appropriate $\boldsymbol{\theta}$. For this, we take the partial derivative of $LL(\theta)$ with respect to $\boldsymbol{\theta}$.

Deriving the Optimization Function

Derive the expression for

$$\frac{\partial LL(\boldsymbol{\theta})}{\partial \theta_j}$$

for $j - th$ component of $\boldsymbol{\theta}$.

Deriving the Optimization Function

Class work

Deriving the Optimization Function

Class work

Deriving the Optimization Function

Class work

Optimization Function

$$\frac{\partial LL(\boldsymbol{\theta})}{\partial \theta_j} = \sum_{i=1}^{n} \Big[y_i - h_{\boldsymbol{\theta}}(\mathbf{x}_i) \Big] x_{ij}$$

The derivation comes from the fact that the logistics function has a special property:

$$\frac{\partial g(z)}{\partial z} = g(z)[1 - g(z)]$$

Logistics Regression with SGD

Equating

$$\frac{\partial LL(\boldsymbol{\theta})}{\partial \theta_j} = 0$$

will give optimal θ_j^*. However, the expression doesn't have any closed form, so we resort to using numerical methods for solving the optimization, and hence, in this case, we use **Gradient Ascent Optimization** which is equivalent to minimizing the negative of the log-likelihood function, for which we can use already discussed **Gradient Descent Optimization**.

Updated Rule

$$\theta_j \leftarrow \theta_j - \eta \frac{\partial LL(\boldsymbol{\theta})}{\partial \theta_j}$$

$$\theta_j \leftarrow \theta_j - \eta \sum_{i=1}^{n} \left[y_i - h_{\theta}(\mathbf{x}_i) \right] x_{ij}$$

Pseudo Code

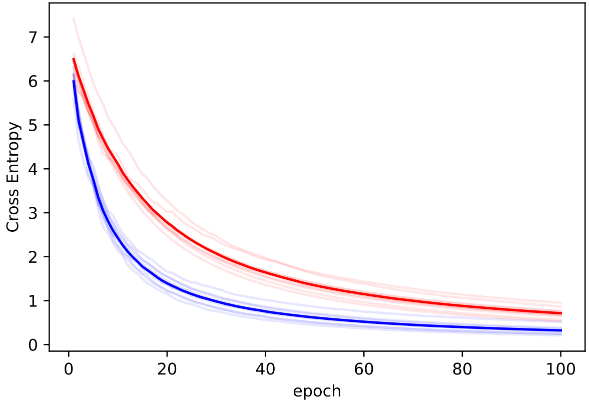
Logistic regression is quite easy to code up using stochastic gradient descent. You'll need an approach to determine when to stop training.

```
**Input**: $\mathcal{D} := \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}, \eta, T$
**Initialize:** $\boldsymbol{\theta} = 0$ and $\theta_0 = 0$        % or initialize from $\mathcal{N}(0, \sigma)$
for $t = 1, \ldots, T$
    for $j = 1, \ldots, n$
        $i = \texttt{np.random.randint}(n)$
        $\boldsymbol{\theta} = \boldsymbol{\theta} + \eta(y_i - g(\mathbf{x}_i))\mathbf{x}_i$
        $\theta_0 = \theta_0 + \eta(y_i - g(\mathbf{x}_i))$
    if CONVERGED, STOP
```

Stopping Criteria

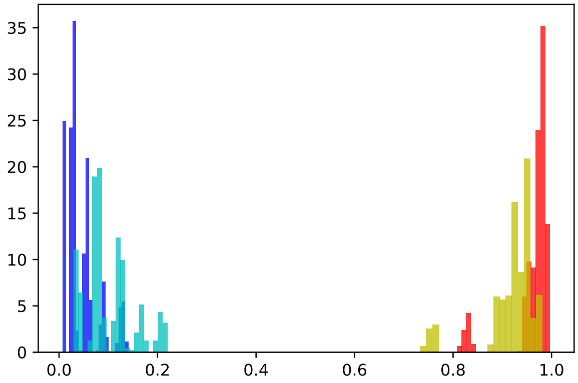
$$\|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\|_2^2 \leq \epsilon, \text{ or } \texttt{CrossEntropy}(t) - \texttt{CrossEntropy}(t+1) \leq \epsilon$$

Logistic Regression Example



Cross entropy loss on the Iris dataset for two logistic regression models. The blue is trained with a learning rate of $\eta = 0.0025$ and the red is trained with a learning rate of $\eta = 0.001$

Logistic Regression Example



Probabilities, $g(\mathbf{x})$, on predicted data on a validation set. The Cyan/Yellow probabilities are from the model with $\eta = 0.001$ and the Blue/Red probabilities are from the model with $\eta = 0.0025$.

Logistic Regression (in code)

```python
class LR:
    def __init__(self, lr=0.0025, epochs=50, split=.1):
        self.lr = lr
        self.epochs = epochs
        self.w = None
        self.b = None
        self.cross_ent = np.zeros(epochs)
        self.split = split

    def score(self, X):
        return 1.0/(1 + np.exp(-(np.dot(self.w, X) + self.b)))
```

Logistic Regression (in code)

```
def crossent(self, X, y):
    ce = np.log(self.score(X[y==1])).sum() + \
              np.log(1.0 - self.score(X[y==0])).sum()
    return -ce
```

Logistic Regression (in code)

```
def fit(self, X, y):
    i = np.random.permutation(len(y))
    X, y = X[i], y[i]
    self.w, self.b = np.zeros(X.shape[1]), 0
    M = np.floor((1-self.split)*len(y)).astype(int)
    Xtr, ytr, Xva, yva = X[:M], y[:M], X[M:], y[M:]
    for t in range(self.epochs):
        # run stochastic gradient descent
        for i in np.random.permutation(len(ytr)):
            self.w += self.lr*(ytr[i] - self.score(Xtr[i]))*Xtr[i]
            self.b += self.lr*(ytr[i] - self.score(Xtr[i]))
        self.cross_ent[t] = self.crossent(Xva, yva)
```

Logistic Regression (in code)

```python
def predict(self, X):
    return 1.0*(self.score(X[i]) >= 0.5)

def predict_proba(self, X):
    return self.score(X[i])

def predict_log_proba(self, X):
    return np.log(self.predict_proba(X))
```

Logistics Regression Using Sklearn

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

# Create an instance of a Logistic Regression Classifier and fit the data.
logreg = LogisticRegression()
logreg.fit(x, y.ravel())

# Predict
y_hat = logreg.predict(x)

print("Predictions:", y_hat)
```

Assessing the Logistic Regression

Accuracy

Accuracy is a metric used to evaluate the performance of classification models. Informally, accuracy is the fraction of predictions our model got right. It measures the ratio of correctly predicted instances (true positives and true negatives) to the total number of instances.

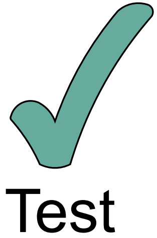
$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Confusion Matrix

Confusion Matrix displays how many of the data points were classified correctly and incorrectly.

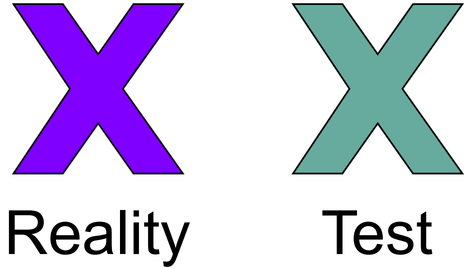
```
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y, y_hat)
print(confusion_matrix)
```

True Positive (tp)



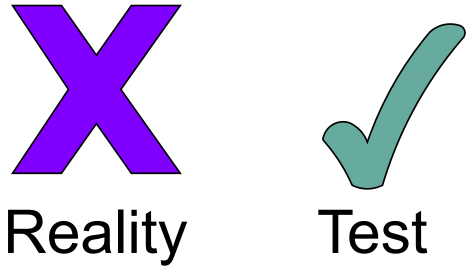
True Positive (tp): Arpita has cancer. She takes a medical test for cancer, and the test result is positive. This is a true positive because the test correctly identified that Arpita has cancer.

True Negative (tn)



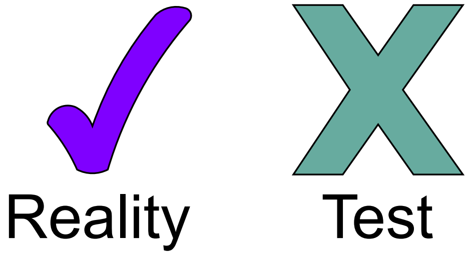
True Negative (tn): Mumtaj does not have cancer. She takes the same medical test, and the test result is negative. This is a true negative because the test correctly identified that Mumtaj does not have cancer.

False Positive (fp)



False Positive (fp): Niu does not have cancer. However, when she takes the medical test, the test result is positive. This is a false positive because the test incorrectly identified that Niu has cancer.

False Negative (fn)



False Negative (fn): Diana has cancer. However, when she takes the medical test, the test result is negative. This is a false negative because the test incorrectly identified that Diana does not have cancer.

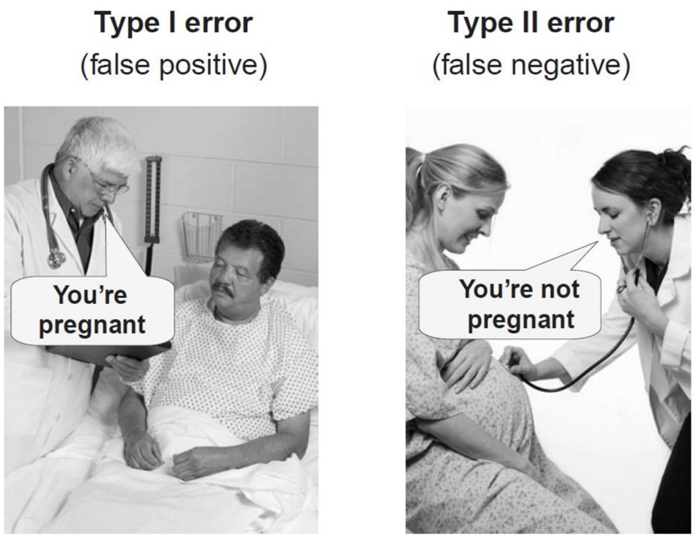


Figure 3.1 Type I and Type II errors

Precision

Precision is the ratio

$$\text{Precision} = \frac{tp}{tp + fp}$$

This means the classifier cannot label a sample as positive if it is negative.

Recall

Recall is

$$\text{Recall} = \frac{tp}{tp + fn}$$

It is the ability of the classifier to find all the positive samples.

F-Score

F-score is the weighted harmonic mean of the precision and recall.

$$F_{score} = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Support

Support is the number of occurrences of each class in the test set.

In Sklearn: ...

```
from sklearn.metrics import classification_report
print(classification_report(y, y_hat))
```

Receiver Operating Characteristic (ROC)

An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds (0.5 by default). This curve plots two parameters:

- True Positive Rate (TPR)
- False Positive Rate (FPR)

An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives.

Classification Threshold

Definition

In binary classification, the threshold determines the decision boundary for predicted probabilities. If the probability is above the threshold, the observation is classified as positive class; below as negative class.

Example

Logistic regression typically uses 0.5 as default threshold. Probabilities ≥ 0.5 are classified as positive, < 0.5 as negative.

Thresholds & ROC Curves

Threshold Variation

Each point on a ROC curve represents a different threshold (0-1). Lower thresholds increase both False Positives and True Positives.

Optimal Threshold

Determined by balancing TP/FP tradeoff. Can be calculated directly from ROC curves or found through grid search, depending on misclassification costs.

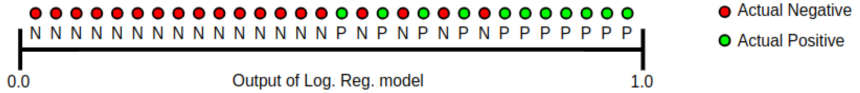
AUC (Area Under the ROC Curve)

Definition

Measures entire area under ROC curve (0-1). Probability that model ranks random positive example higher than random negative example.

Interpretation

0.0 = 100% wrong predictions
1.0 = 100% correct predictions



N N N N N N N N N N N N N N N P N P N P N P N P P P P P P P
0.0
Output of Log. Reg. model
1.0
Actual Negative
Actual Positive

AUC Properties & Caveats

Advantages

- Scale-invariant: Ranks predictions well
- Threshold-invariant: Evaluates all thresholds

Limitations

- Doesn't assess calibrated probabilities
- Not suitable for cost-sensitive tasks

AUC using Sklearn

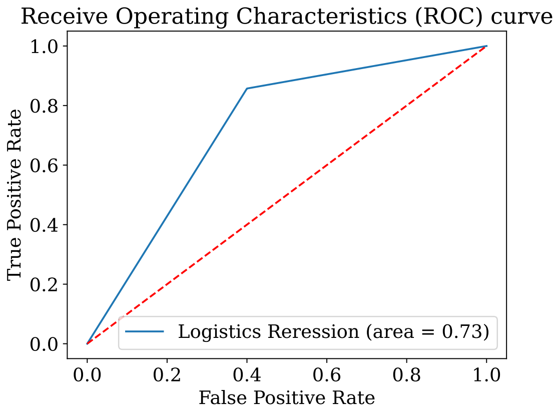
```python
# Example from sklearn documentation
from sklearn.metrics import roc_auc_score, roc_curve
y = [1, 1, 0, 0]
y_hat = [0.9, 0.6, 0.4, 0.2]
print(roc_auc_score(y, y_hat))  # Output: 1.0
```

ROC in Sklearn

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y, y_hat)
fpr, tpr, threshold = roc_curve(y, y_hat)

plt.figure()
plt.plot(fpr, tpr,
    label='Logistics Reression (area = %0.2f)'% logit_roc_auc)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receive Operating Characteristics (ROC) curve')
plt.legend(loc='lower right')
plt.show()
```

ROC in Sklearn



Multi-Class Logistic Regression

Multi-class

The binary logistic regression assumes that the label $y_i \in \{0, 1\}$, while multi-class cases have more than two classes, i.e. , $y_i \in \{1, 2, \cdots, k\}$ where k is the number of classes.

- In this case, we need to estimate the probability for each of the K classes.

Hypothesis in Multi-class

Multi-class hypothesis is written as

$$h_\theta = \begin{bmatrix} P(y = 1|\mathbf{x};\boldsymbol{\theta}) \\ P(y = 2|\mathbf{x};\boldsymbol{\theta}) \\ \cdots \\ P(y = k|\mathbf{x};\boldsymbol{\theta}) \end{bmatrix}$$

Assumption: no ordinal relationship between classes.

Hypothessis in Multi-class

We will need one linear signal for each of the k classes, which should be independent conditioned on $\mathbf{x}$.
We compute k linear signals by the dot product between the input x and k independent weight vectors $\mathbf{w}_k$ as

$$\begin{bmatrix} \mathbf{w}_1^\top \mathbf{x} \\ \mathbf{w}_2^\top \mathbf{x} \\ \mathbf{w}_k^\top \mathbf{x} \end{bmatrix}$$

Now, we need to map k linear outputs to k probability.

Softmax Function

We need to use softmax function for generalized form the signmoid function which can be written as:

$$softmax(y_i) = \frac{\exp(y_i)}{\sum_{j=1}^{n} \exp(y_j)}$$

Softmax function gives the probability of each sample belonging to a certain class.
Thus,

$$h_\theta = \frac{1}{\sum_{i=1}^{k} \exp\left(\mathbf{w}_i^\top \mathbf{x}\right)} \begin{bmatrix} \exp\left(\mathbf{w}_1^\top \mathbf{x}\right) \\ \exp\left(\mathbf{w}_2^\top \mathbf{x}\right) \\ \cdots \\ \exp\left(\mathbf{w}_k^\top \mathbf{x}\right) \end{bmatrix}$$

Loss Function in Multi-class Logistics Regression

$$L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta}) = -\sum_k \mathbf{y}_{ik} \log \hat{\mathbf{y}}_{ik} = -\sum_k \mathbf{y}_{ik} \log \left(\frac{\exp\left(\boldsymbol{\theta}_k^\top \mathbf{x}_i\right)}{\sum_j^k \exp\left(\boldsymbol{\theta}_j^\top \mathbf{x}_i\right)} \right)$$

y_i is one-hot encoding for labels, $\hat{\mathbf{y}}_i$ is the predicted distribution.

One Hot Encoding for Multi-class Scenario

Consider four classes: {Dog, Cat, Horse, Cheetah } that can be assigned numerals as {1, 2, 3, 4}. In one hot-encoding, it can be written as:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & -- & Dog \\ 0 & 1 & 0 & 0 & -- & Cat \\ 0 & 0 & 1 & 0 & -- & Horse \\ 0 & 0 & 0 & 1 & -- & Cheetah \end{bmatrix}$$

Code Availability and Python Notebook

Code used for this chapter is available at
https://github.com/rahulbhadani/CPE490_590_Sp2025/blob/master/Code/Chapter_08_Logistics_Regression.ipynb

The End