Introduction

Data scientists are often tasked with automating decisions for business problems. Is an email an attempt at phishing? Is a customer likely to churn? Is the web user likely to click on an advertisement? These are all classification problems, a form of supervised learning in which we first train a model on data where the outcome is known and then apply the model to data where the outcome is not known.

Classification is
perhaps the most important form of prediction: the goal is to predict whether a record is a 1 or a 0 (phishing/not-phishing, click/don't click, churn/don't churn), or in some cases, one of several categories (for example, Gmail's filtering of your inbox into "primary," "social," "promotional," or "forums").

Often, a simple yes/no answer is not enough. Instead of only asking *"Does this email belong to the phishing class?"*, we may want to know **how likely** it is to be phishing. For example, an email might have a 90% probability of being phishing, while another has only 55%. This extra information is very valuable for decision-making.

Rather than forcing a strict binary decision, most classification algorithms can output a **probability score** (also called a *propensity*) that indicates how strongly an observation belongs to a given class. For instance, in customer churn prediction, a customer with a churn probability of 0.8 may be treated very differently from one with a probability of 0.3.

In **logistic regression**, the model naturally works on **log-odds**. In R, the default output is on this log-odds scale, so it must be converted into a probability between 0 and 1 to be easily interpreted.

In **Python's scikit-learn**, logistic regression (like most classifiers) offers two ways to make predictions:

- `predict()` returns the final class (e.g., churn or not churn).
- `predict_proba()` returns the probability for each class (e.g., 70% chance of churn).

Once we have probabilities, we can apply a **sliding cutoff** to make decisions. For example, instead of using the default threshold of 0.5, a bank might flag emails as phishing only if the probability is above 0.9, reducing false alarms, or lower the threshold to catch more risky cases. This flexibility is a major advantage of probabilistic classification.

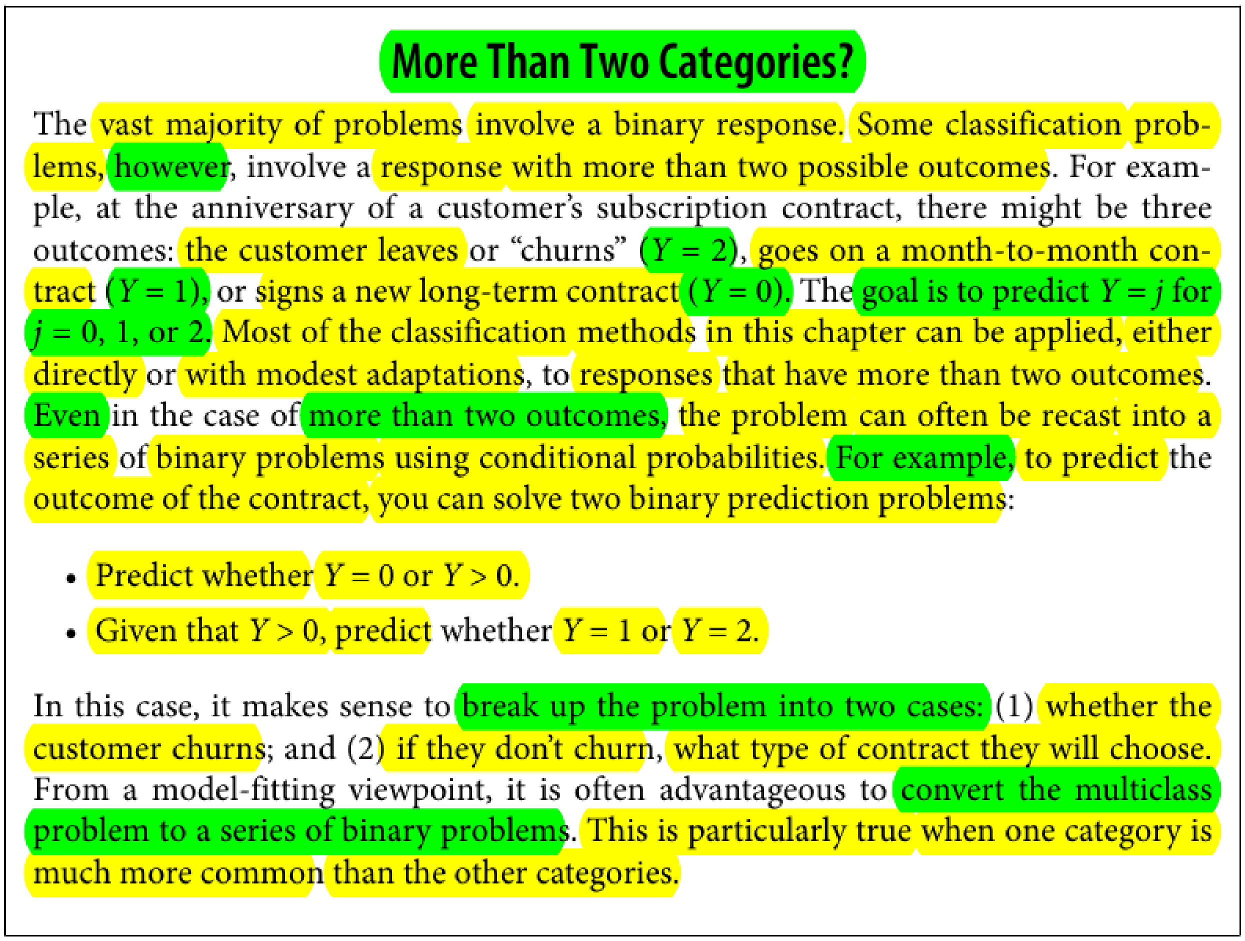
The general approach is as follows:

1. Set cutoff probability:reflects how aggressive we want to be. For example, a company might decide that only customers with a churn probability above **0.8** are worth targeting, because retention campaigns are costly.

2. Next, we **use a model to estimate probabilities**. Any classification model that outputs probabilities can be used—such as logistic regression or a tree-based model. For instance, the model may predict that Customer A has a **0.85** probability of churning, while Customer B has only **0.40**

3.Finally, we **compare the predicted probability to the cutoff**. If the probability exceeds the cutoff, we assign the record to the class of interest. In our example, Customer A would be classified as “likely to churn” and receive a retention email, while Customer B would not.

The higher the cutoff, the fewer the records predicted as 1 the class of interest.
The lower the cutoff, the more the records predicted as 1.



More Than Two Categories?

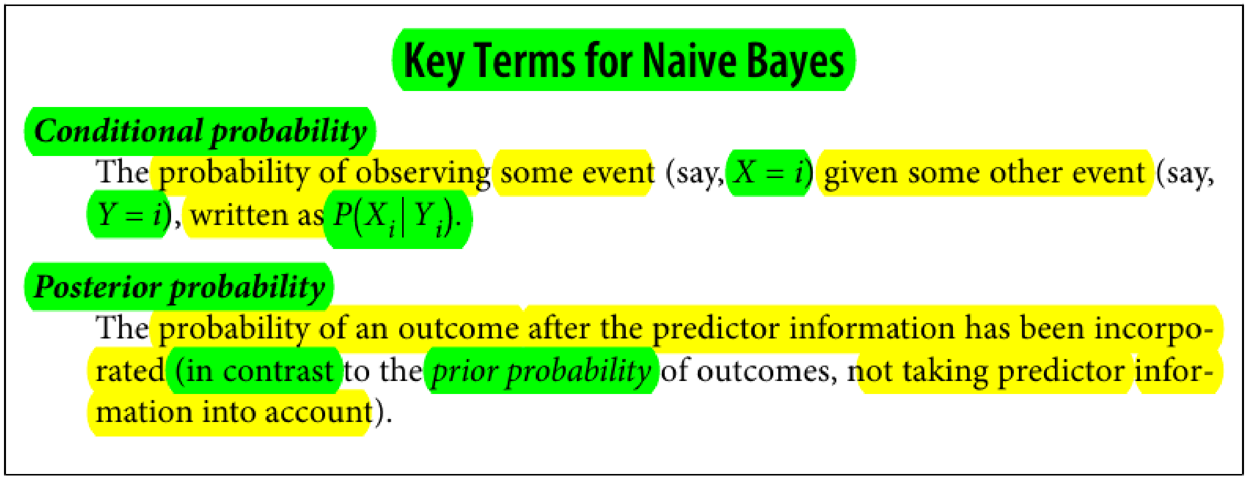
The vast majority of problems involve a binary response. Some classification problems, however, involve a response with more than two possible outcomes. For example, at the anniversary of a customer’s subscription contract, there might be three outcomes: the customer leaves or “churns” ($Y = 2$), goes on a month-to-month contract ($Y = 1$), or signs a new long-term contract ($Y = 0$). The goal is to predict $Y = j$ for $j = 0, 1, \text{ or } 2$. Most of the classification methods in this chapter can be applied, either directly or with modest adaptations, to responses that have more than two outcomes. Even in the case of more than two outcomes, the problem can often be recast into a series of binary problems using conditional probabilities. For example, to predict the outcome of the contract, you can solve two binary prediction problems:

- Predict whether $Y = 0$ or $Y > 0$.
- Given that $Y > 0$, predict whether $Y = 1$ or $Y = 2$.

In this case, it makes sense to break up the problem into two cases: (1) whether the customer churns; and (2) if they don’t churn, what type of contract they will choose. From a model-fitting viewpoint, it is often advantageous to convert the multiclass problem to a series of binary problems. This is particularly true when one category is much more common than the other categories.

I.Naive Bayes

The key idea behind the **Naive Bayes** algorithm is that it starts from probabilities that are easy to measure and uses them to estimate the probability we actually care about. Naive Bayes uses *what we can easily observe* (predictors given an outcome Y = i) to infer *what we really want to know* (the outcome given the predictors).



Key Terms for Naive Bayes

Conditional probability

The probability of observing some event (say, $X = i$) given some other event (say, $Y = i$), written as $P(X_i \mid Y_i)$.

Posterior probability

The probability of an outcome after the predictor information has been incorporated (in contrast to the *prior probability* of outcomes, not taking predictor information into account).

To understand naive Bayesian classification, we can start out by imagining complete or exact Bayesian classification. For each record to be classified:

1. Suppose we want to classify a new record, such as an email or a customer profile. First, we would look through our dataset and **find all past records with exactly the same predictor values**. For an email, this would mean having the same words, same sender features, and same metadata.

2. Next, we would **check which classes those matching records belong to**. For example, among all emails with that exact profile, some may be phishing and others legitimate.

3. Finally, we would **assign the most common class** among those records to the new one. If most of the identical past emails were phishing, we would label the new email as phishing as well.

The preceding approach amounts to finding all the records in the sample that are exactly like the new record to be classified in the sense that all the predictor values are identical.

Why Exact Bayesian Classification Is Impractical ?

Exact Bayesian classification quickly becomes impractical as the number of predictors grows. When there are more than just a few variables, the chance of finding past records with **exactly the same combination of values** becomes very small.

For example, imagine predicting how someone will vote using demographic data. Even with a large dataset, it is unlikely to contain an exact match for a person who is a Hispanic male, high income, living in the U.S. Midwest, who voted in the last election but not the one before, and has a specific family situation. This already involves only eight variables, which is relatively small for real classification tasks.

The problem gets worse as more variables are added. Introducing just **one additional variable** with five equally common categories divides the chance of an exact match by five. As a result, exact matching becomes nearly impossible, which is why simpler probabilistic approaches like Naive Bayes are needed.

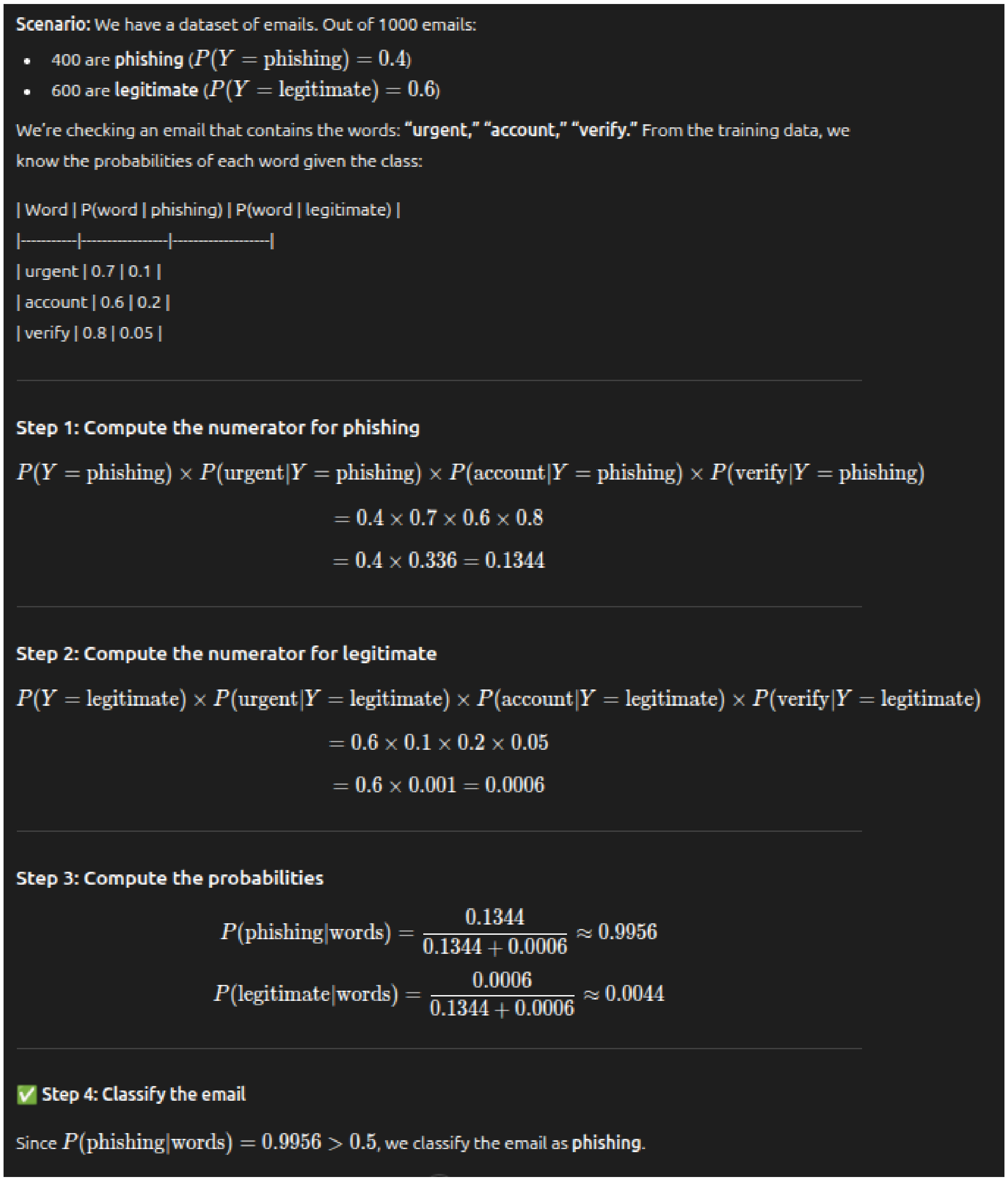
I.1 The Naive Solution

The **naive Bayes solution** avoids the impracticality of exact matching by using the **entire dataset** instead of looking for perfect matches. Here's how it works, step by step:

1. For each predictor Xj and each class Y=i (e.g., 0 or 1), calculate the **conditional probability** P(Xj∣Y=i). This is just the proportion of times that predictor value appears among records of class iii. For example, if 70% of phishing emails contain the word “urgent,” then P(“urgent”∣phishing)=0.7.
2. **Multiply these probabilities together** for all predictors and then multiply by the overall proportion of records in class iii (the prior probability P(Y=i)P(Y = i)P(Y=i)).
3. **Repeat steps 1 and 2 for all classes**.
4. **Normalize**: divide the value for class iii by the sum of the values for all classes. This gives the probability of the record belonging to class iii.
5. **Assign the class** with the highest probability to the record.

This means we **treat each predictor as independent given the class**, making the calculation practical while still producing surprisingly accurate results.

Example: For a phishing email with the words “urgent,” “account,” and “verify,” we multiply the probabilities of seeing each word in phishing emails, multiply by the overall chance of an email being phishing, and compare to the same calculation for legitimate emails. The class with the higher probability wins.



Scenario: We have a dataset of emails. Out of 1000 emails:

- 400 are **phishing** ($P(Y = \text{phishing}) = 0.4$)
- 600 are **legitimate** ($P(Y = \text{legitimate}) = 0.6$)

We're checking an email that contains the words: **"urgent," "account," "verify."** From the training data, we know the probabilities of each word given the class:

Word	P(word | phishing)	P(word | legitimate)
urgent	0.7	0.1
account	0.6	0.2
verify	0.8	0.05

Step 1: Compute the numerator for phishing

$$P(Y = \text{phishing}) \times P(\text{urgent}|Y = \text{phishing}) \times P(\text{account}|Y = \text{phishing}) \times P(\text{verify}|Y = \text{phishing})$$
$$= 0.4 \times 0.7 \times 0.6 \times 0.8$$
$$= 0.4 \times 0.336 = 0.1344$$

Step 2: Compute the numerator for legitimate

$$P(Y = \text{legitimate}) \times P(\text{urgent}|Y = \text{legitimate}) \times P(\text{account}|Y = \text{legitimate}) \times P(\text{verify}|Y = \text{legitimate})$$
$$= 0.6 \times 0.1 \times 0.2 \times 0.05$$
$$= 0.6 \times 0.001 = 0.0006$$

Step 3: Compute the probabilities

$$P(\text{phishing}|\text{words}) = \frac{0.1344}{0.1344 + 0.0006} \approx 0.9956$$
$$P(\text{legitimate}|\text{words}) = \frac{0.0006}{0.1344 + 0.0006} \approx 0.0044$$

✅ **Step 4: Classify the email**

Since $P(\text{phishing}|\text{words}) = 0.9956 > 0.5$, we classify the email as **phishing**.

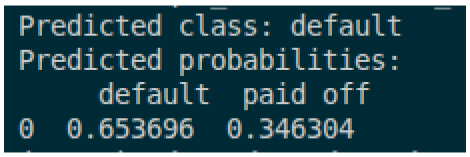
The formula is called **"naive"** because it makes a very strong—and obviously unrealistic—assumption: it treats each predictor as **independent of the others** given the class.

In reality, predictors often influence each other. For example, in an email, the presence of the word **"account"** might make **"verify"** more likely, so they are not truly independent.

Naive Bayes **ignores these relationships** and estimates the joint probability of all predictors as the **product of the individual probabilities.**

In Python we can use sklearn.naive_bayes.MultinomialNB from scikit-learn. We need to convert the categorical features to dummy variables before we fit the model.

As we discussed, scikit-learn's classification models have two methods—predict, which returns the predicted class, and predict_proba, which returns the class probabilities.



```
Predicted class: default
Predicted probabilities:
     default  paid off
0  0.653696  0.346304
```

Predicted class: default :
The model's final decision is that this loan (row 146 arbitrarily choosed) is most likely to **default**. *Naive Bayes assigns a record to the class with the highest probability by default.*

```
      default   paid off
0    0.653696   0.346304
```

These numbers are the probabilities that this loan belongs to each class, given its features (purpose_, home_, emp_len_):

0.653696 → probability of default ≈ 65%
0.346304 → probability of paid off ≈ 35%

This tells us that, according to the model, this loan is more likely to default than to be fully paid,

steps

- The Naive Bayes model looks at the predictors (loan purpose, home ownership, employment length) and uses the probabilities learned from past data.
- It multiplies the conditional probabilities for each predictor (naively assuming independence), multiplies by the prior probability of each class, and **normalizes** to get these probabilities.
- The class with the **highest probability wins**. Here, `default` wins because 65% > 35%.

I.2 Numeric Predictor Variables

The Bayesian classifier works only with categorical predictors (e.g., with spam classi-
fication, where the presence or absence of words, phrases, characters, and so on lies at
the heart of the predictive task). To apply naive Bayes to numerical predictors, one of
two approaches must be taken:

• Bin and convert the numerical predictors to categorical predictors and apply the algorithm of the previous section.

• Use a probability model—for example, the normal distribution (see "Normal Distribution" on page 69)—to estimate the conditional probability P X j Y = i .

When a predictor category is absent in the training data, the algorithm assigns *zero probability* to the outcome variable in new data, rather than simply ignoring this variable and using the information from other variables, as other methods might. Most implementations of Naive Bayes use a smoothing parameter (Laplace Smoothing) to prevent this.

Scenario:

We want to predict whether a loan **defaults** (`Y=1`) or is **paid off** (`Y=0`) based on a single categorical predictor: **loan purpose**.

purpose	**outcome**
car	0
car	1
home	0
major	1

We encode the probabilities from the training set:

- `P(car | default=1)` → 1 record out of 2 defaults has purpose “car” → **0.5**
- `P(home | default=1)` → 0 records out of 2 defaults → **0**
- `P(major | default=1)` → 1 record out of 2 defaults → **0.5**

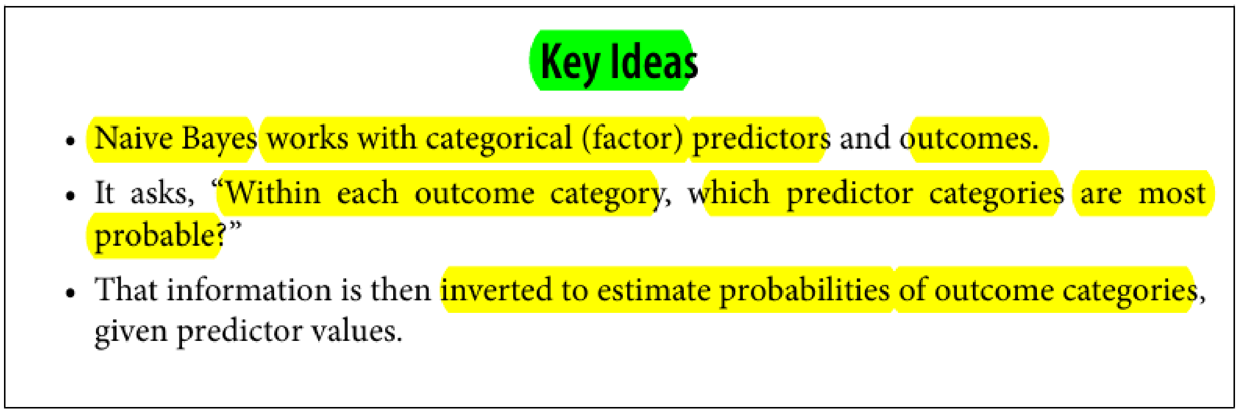
Problem:

Now we get a **new loan** with `purpose = "vacation"` (a category **not seen in training**). But `P(purpose="vacation"|Y=1) = 0` (never appeared in training).

Similarly, `P(purpose="vacation"|Y=0) = 0` also be 0 if “vacation” never appeared for non-defaults.

✅ Result: The algorithm **breaks** — it assigns zero probability to all outcomes.

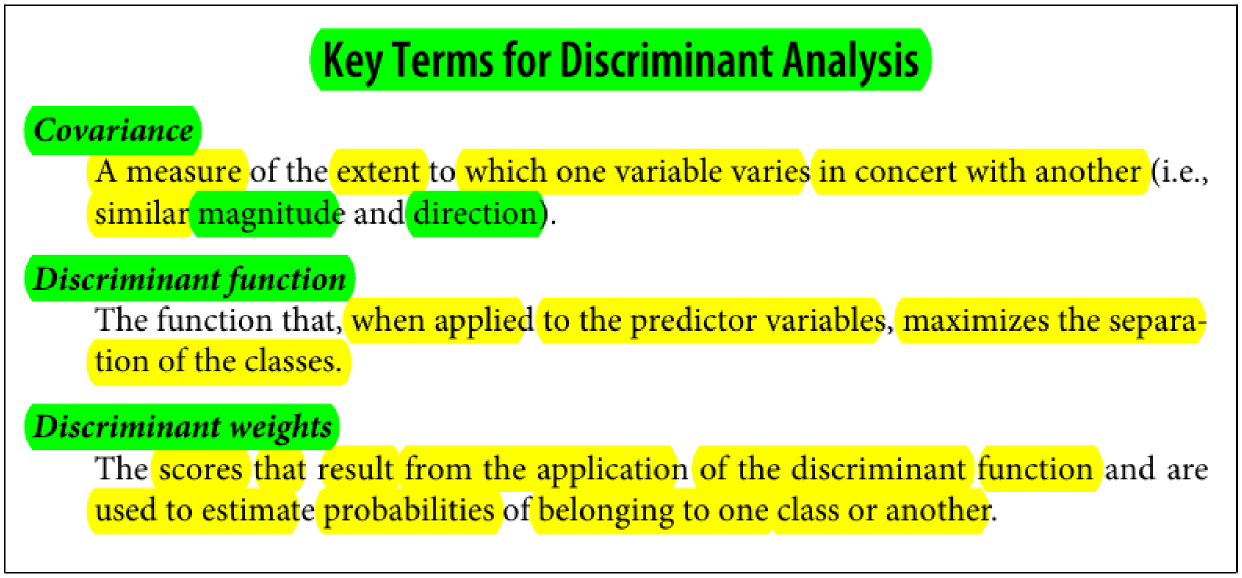
Class is the outcome (default/ not paid off)



Key Ideas

- Naive Bayes works with categorical (factor) predictors and outcomes.
- It asks, "Within each outcome category, which predictor categories are most probable?"
- That information is then inverted to estimate probabilities of outcome categories, given predictor values.

II. Discriminant Analysis



Key Terms for Discriminant Analysis

Covariance
A measure of the extent to which one variable varies in concert with another (i.e., similar magnitude and direction).

Discriminant function
The function that, when applied to the predictor variables, maximizes the separation of the classes.

Discriminant weights
The scores that result from the application of the discriminant function and are used to estimate probabilities of belonging to one class or another.

It was originally proposed by **Fisher**, and while his method was slightly different, the mechanics are very similar to what we now call LDA. The idea is to **find a linear combination of predictors that best separates the classes**. For example, if you want to classify emails as phishing or legitimate based on features like word counts or sender reputation, LDA tries to find a line (or hyperplane in higher dimensions) that best separates the two groups.

LDA is **less popular today** because modern techniques like **tree-based models** and **logistic regression** are often more flexible and accurate, especially when relationships are nonlinear.

However, LDA still shows up in some applications, and it has interesting connections to other methods, like **Principal Components Analysis (PCA)**, which also looks for linear combinations of variables—but to capture **variance** instead of class separation.

II.1 Covariance Matrix

Before we can understand **discriminant analysis**, we need to understand **covariance**, because LDA and similar techniques rely on how variables relate to each other.

Covariance measures how two variables, say x and z, **change together**:

- If both tend to increase together, the covariance is **positive**.
- If one tends to increase when the other decreases, the covariance is **negative**.
- If they are unrelated, the covariance is near **zero**.

Student	**Math ((x))**	**Physics ((z))**
1	80	85
2	90	95
3	70	75

Positive covariance → as Math scores increase, Physics scores also tend to increase.

Important: Because covariance depends on units, we often prefer the **correlation coefficient** (−1 to +1) to judge strength.

Step 1 : data

Student	**Math (x)**	**Physics (z)**
1	70	80
2	85	90
3	90	75

Step 2: Means
- Mean of Math ($\bar{x}$) = 81.67
- Mean of Physics ($\bar{z}$) = 81.67

Step 3: Variances
- Variance of Math (sx^2) = 108.3
- Variance of Physics (sz^2) = 58.35

Step 4: Covariance
- Covariance between Math and Physics (sx,z) = -4.15

  Interpretation: Negative → Math and Physics tend to move slightly in opposite directions.

Step 5: Covariance Matrix

	Math (x)	**Physics (z)**
Math (x)	108.3	-4.15
Physics (z)	-4.15	58.35

Recall that the standard deviation is used to normalize a variable to a z-score; the covariance matrix is used in a multivariate extension of this standardization process (When you have **more than one variable** (like Math and Physics scores), you need a **multivariate version of standardization**. Here, each variable has its variance (spread) and covariances with other variables.)

A **multivariate extension of standardization** uses **both variances and covariances.**

II.2 Fisher's Linear Discriminant

Fisher's LDA aims to:

1. **Maximize the separation between groups** → "between-group variance"
2. **Minimize the spread within each group** → "within-group variance"

Mathematically, it finds a **linear combination** of the predictors x and z:

$$y_{\text{score}} = w_x \cdot x + w_z \cdot z$$

where the weights wx and wz are chosen to **maximize the ratio of between-group variance to within-group variance**:

$$\text{maximize } \frac{SS_{\text{between}}}{SS_{\text{within}}}$$

- SS_{between} → squared distance between group means
- SS_{within} → spread of points inside each group

Step 1: Recall the data

Student	Math (x)	Physics (z)	Outcome (y)
1	70	80	0
2	85	90	1
3	90	75	1

Step 2: Assume LDA finds the weights

- wx=0.6
- wz=0.4

(In practice, these weights are computed from group means and covariance.)

Step 3: Compute the LDA score for each student

- **Student 1:** 0.6 × 70 + 0.4 × 80 = 42 + 32 = 74
- **Student 2:** 0.6 × 85 + 0.4 × 90 = 51 + 36 = 87

- **Student 3:** 0.6 × 90 + 0.4 × 75 = 54 + 30 = 84

Step 4: Determine a threshold

- Group 0 (fail) score: 74
- Group 1 (pass) scores: 87, 84

$$\text{threshold} = \frac{74 + \frac{87+84}{2}}{2} = \frac{74 + 85.5}{2} \approx 79.75$$

Threshold (midway between group means):

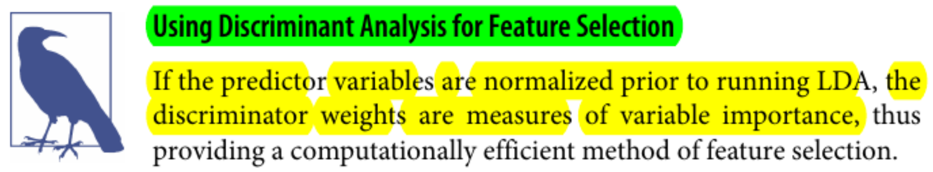
- Score > 79.75 → predict **pass (y=1)**
- Score ≤ 79.75 → predict **fail (y=0)**

Step 5: Apply threshold

Student	LDA score	Predicted y
1	74	0 (fail)
2	87	1 (pass)
3	84	1 (pass)

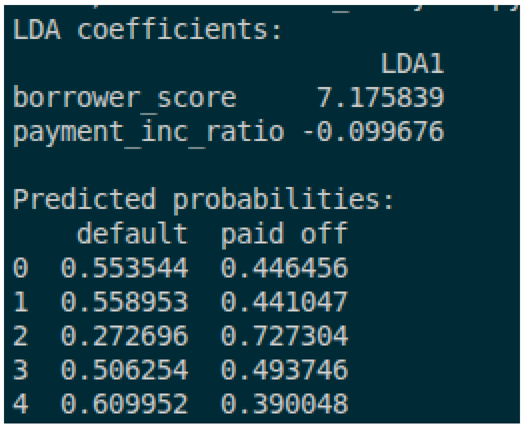
II.2 a simple example

In Python, we can use LinearDiscriminantAnalysis from sklearn.discriminant_analysis. The scalings_ property gives the estimated weights.



Using Discriminant Analysis for Feature Selection

If the predictor variables are normalized prior to running LDA, the discriminator weights are measures of variable importance, thus providing a computationally efficient method of feature selection.



```
LDA coefficients:
                         LDA1
borrower_score     7.175839
payment_inc_ratio -0.099676

Predicted probabilities:
    default  paid off
0  0.553544  0.446456
1  0.558953  0.441047
2  0.272696  0.727304
3  0.506254  0.493746
4  0.609952  0.390048
```

1 LDA coefficients (this *is* Fisher's linear discriminant)

`LDA1 = 7.175839 × borrower_score - 0.099676 × payment_inc_ratio`

Variable	Coefficient	Interpretation
borrower_score	7.18	Strong positive effect → higher score pushes toward paid off
payment_inc_ratio	−0.10	Negative effect → higher burden pushes toward default

- **`borrower_score`** dominates the decision
- **`payment_inc_ratio`** matters, but much less

LDA score = 7.18 × borrower_score − 0.10 × payment_inc_ratio

2 0.2727 0.7273 : *LDA thinks this borrower is much more likely to pay off.*

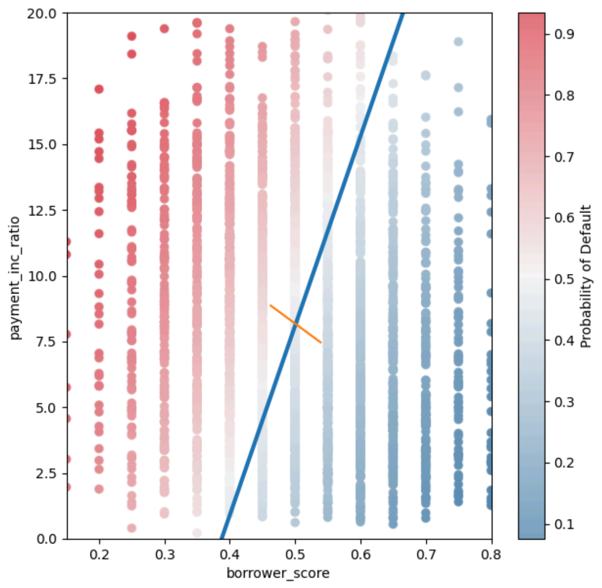


Figure 5-1. LDA prediction of loan default using two variables: a score of the borrower's creditworthiness and the payment-to-income ratio.

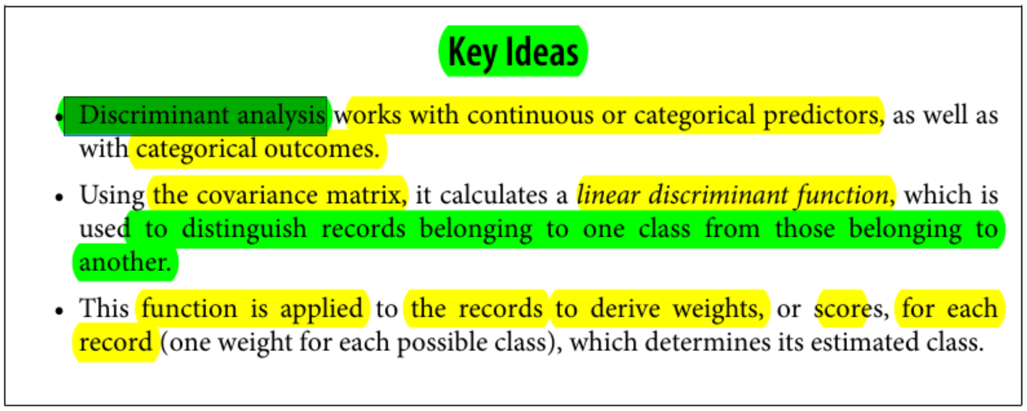
Using the discriminant function weights, LDA splits the predictor space into two regions, as shown by the solid line.

Extensions of Discriminant Analysis

Linear Discriminant Analysis (LDA) isn't limited to just two predictor variables. You can include many predictors, and LDA still works well. The main practical constraint is the number of records you have, because estimating the covariance matrix requires enough data for each variable—but in most data science applications, this isn't a problem.

There are also other variants of discriminant analysis. The most well-known is **Quadratic Discriminant Analysis (QDA)**. Despite its name, QDA still produces a linear discriminant function, but with a key difference: in LDA, we assume the covariance matrix is the same for both groups (e.g., Y=0Y=0Y=0 and Y=1Y=1Y=1), whereas QDA allows each group to have its own covariance matrix. In practice, this difference often doesn't make a huge impact on results, but QDA can be helpful if the groups truly have very different spreads or correlations among variables.

Example:
Suppose we're classifying emails as spam or not spam based on word counts. With LDA, we assume the variance and correlation of word counts are the same for both spam and non-spam emails. With QDA, we allow spam emails to have different variability in word counts than non-spam emails. In many cases, LDA is sufficient, but QDA could capture subtle differences if the data warrants it.



Key Ideas

- Discriminant analysis works with continuous or categorical predictors, as well as with categorical outcomes.
- Using the covariance matrix, it calculates a *linear discriminant function*, which is used to distinguish records belonging to one class from those belonging to another.
- This function is applied to the records to derive weights, or scores, for each record (one weight for each possible class), which determines its estimated class.

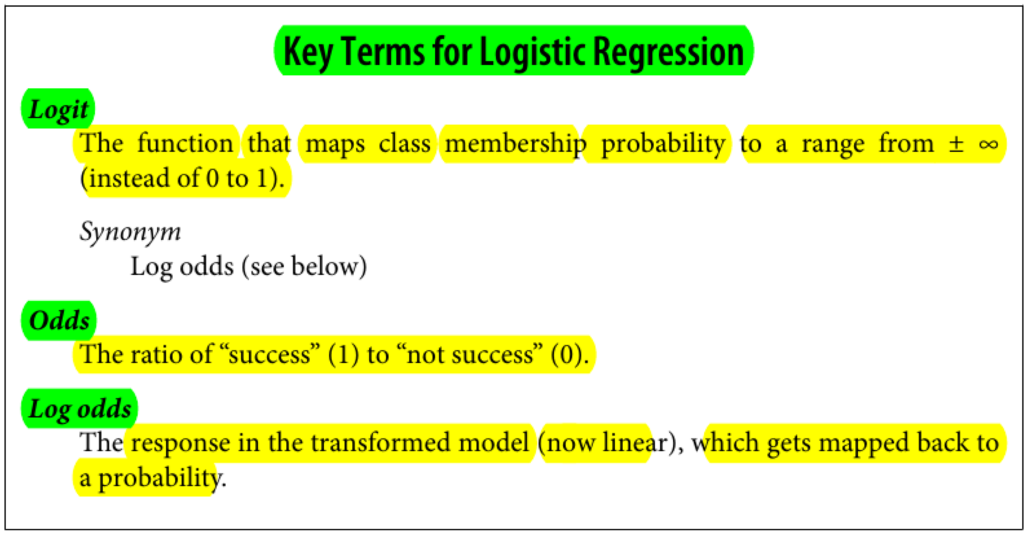
III. Logistic Regression

Logistic regression is similar to multiple linear regression, but it's used when the outcome is **binary**—for example, yes/no, spam/not spam, or sick/healthy. Various transformations are employed to convert the problem to one in which a linear model can be fit.

Like discriminant analysis, logistic regression is a **structured model**—it assumes a specific relationship between predictors and the outcome—unlike K-Nearest Neighbor or Naive Bayes, which are more data-centric approaches.

One of its strengths is **speed**: once the model is trained, it can quickly score new observations, making it very practical in applications like credit scoring or email spam detection.

Example:
Imagine predicting whether a customer will buy a product based on age and income. Logistic regression fits a model that estimates the probability of purchase for any combination of age and income, allowing you to classify customers as likely or unlikely to buy.



Key Terms for Logistic Regression

Logit
The function that maps class membership probability to a range from ± ∞ (instead of 0 to 1).

Synonym
Log odds (see below)

Odds
The ratio of "success" (1) to "not success" (0).

Log odds
The response in the transformed model (now linear), which gets mapped back to a probability.

III.1. Logistic Response Function and Logit

Logistic regression relies on two main ideas: the **logistic function** and the **logit**.

- The **logistic function** transforms any input into a probability between 0 and 1.
- The **logit** is the logarithm of the odds, which maps a probability (0–1) onto a scale from −∞-\infty−∞ to +∞+\infty+∞, making it suitable for **linear modeling**.

The first step in logistic regression is to think of the outcome not just as a binary label (0 or 1) but as the **probability p** that the outcome is 1 ‘1 represents an event of intrest’.

A naive approach would be to model p as a linear combination of predictors:

$$p = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_q x_q$$

However, this doesn’t work well because a linear model can produce values **less than 0 or greater than 1**, which aren’t valid probabilities.Instead, we model p by applying a logistic response or inverse logit function to the predictors:

$$p = \frac{1}{1 + e^{-\left(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_q x_q\right)}}$$

This transform ensures that the p stays between 0 and 1.

From Probabilities to Log-Odds in Logistic Regression

Probabilities are always between 0 and 1, which makes them tricky to model directly with a linear equation. To fix this, logistic regression uses **odds** instead.

- **Odds** measure the ratio of “successes” (1) to “failures” (0).

$$\text{Odds}(Y = 1) = \frac{p}{1 - p}$$

Example: If a horse has a 0.5 probability of winning, the odds are 0.5/(1−0.5)=1

Next, We combine this with the logistic response function, shown earlier, to get:

$$\text{Odds}(Y = 1) = e^{\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_q x_q}$$

Finally, taking the **logarithm of the odds** gives a linear relationship:

$$\log\left(\text{Odds}(Y = 1)\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_q x_q$$

This is called the **logit function**. It transforms probabilities from 0–1 into the **entire real line** (−∞,+∞), which lets us use a linear model to predict them.

Example:
Predicting whether a student passes based on study hours:

- More hours → higher probability → higher odds → higher log-odds
- After predicting probability, we apply a **cutoff** (e.g., 0.5) to classify:
    - p>0.5→p > 0.5 \rightarrowp>0.5→ predict pass (1)
    - p≤0.5→p \le 0.5 \rightarrowp≤0.5→ predict fail (0)

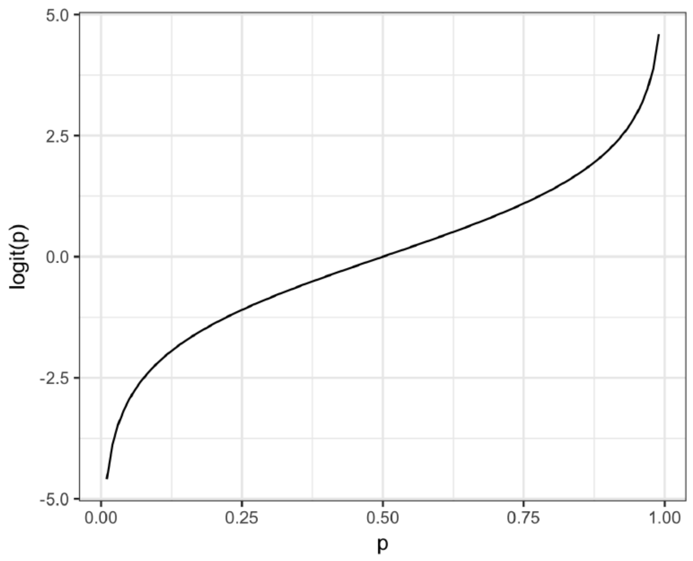
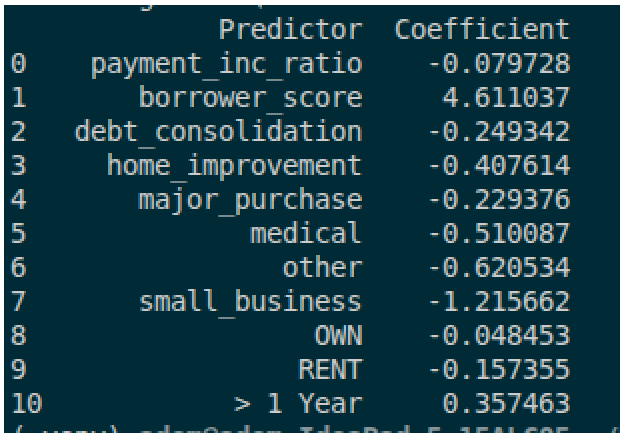


Figure 5-2. Graph of the logit function that maps a probability to a scale suitable for a linear model

III.2. Logistic Regression and the GLM

In logistic regression, the response variable in the formula is the **log-odds** of the outcome being 1. But in reality, we **only observe the binary outcomes** (0 or 1), not the log-odds themselves. Because of this, we can't use ordinary linear regression methods — we need **special statistical techniques** to estimate the model parameters.



```
             Predictor  Coefficient
0    payment_inc_ratio    -0.079728
1       borrower_score     4.611037
2   debt_consolidation    -0.249342
3     home_improvement    -0.407614
4       major_purchase    -0.229376
5              medical    -0.510087
6                other    -0.620534
7       small_business    -1.215662
8                  OWN    -0.048453
9                 RENT    -0.157355
10            > 1 Year     0.357463
```

The response is outcome, which takes a 0 if the loan is paid off and a 1 if the loan defaults.

purpose_ and home_ are factor variables representing the purpose of the loan and the home ownership status. As in linear regression, a factor variable with P levels is represented with P – 1 columns. By default in R, the reference coding is used, and the levels are all compared to the reference level (see “Factor Variables in Regression” on page 163).

The reference levels for these factors are credit_card and MORTGAGE, respectively. The variable borrower_score is a score from 0 to 1 representing the creditworthiness of the borrower (from poor to excellent). This variable was created from several other variables using K-Nearest Neighbor—see “KNN as a Feature Engine” on page 247.

In Python, we use the scikit-learn class LogisticRegression from sklearn.linear_model. The arguments penalty and C are used to prevent overfitting by L1 or L2 regularization. Regularization is switched on by default. In order to fit without regularization, we set C to a very large value. The solver argument selects the used minimizer; the method liblinear is the default.

By default, scikit-learn orders these classes **alphabetically**.

- **Example:** If your classes are `"default"` and `"paid off"`, scikit-learn will internally treat `"default"` as 0 and `"paid off"` as 1.
- logit_reg.classes_   # Output: ['default', 'paid off']
- logit_reg.predict(X)   # Returns class labels for each observation
- logit_reg.predict_proba(X) # Returns probabilities for ['default', 'paid off']

III.3. Generalized Linear Models

GLMs extend linear regression to handle a wider variety of **response types**. They are defined by two key components:

1. **Probability distribution (family):**
   - Determines the type of response variable.
   - **Example:** Logistic regression uses the **binomial family** for binary outcomes. Poisson distribution is used for **count data** (e.g., number of times a user visits a website), and gamma or negative binomial can model **time or duration** (e.g., time to machine failure).
2. **Link function:**
   - Transforms the expected value of the response to a linear combination of predictors.
   - **Example:** Logistic regression uses the **logit link** to map probabilities to the real line. Sometimes a **log link** is used instead, which often gives similar results in practice.

Example in practice:

- Predicting **loan default** → binary outcome → binomial family + logit link → standard logistic regression.

- Predicting **daily website visits** → count outcome → Poisson family + log link → Poisson regression.

log link function is different from the logit. Instead of taking the log of the odds, it takes the **log of the expected value** of the response.

Logistic regression is by far the most common form of **GLM**. A data scientist will encounter other types of **GLMs**. Sometimes a **log link function** is used instead of the **logit**; in practice, use of a **log link** is unlikely to lead to very different results for most applications. The **Poisson distribution** is commonly used to model **count data** (e.g., the number of times a user visits a web page in a certain amount of time). Other families include **negative binomial** and **gamma**, often used to model **elapsed time** (e.g., time to failure). In contrast to **logistic regression**, application of **GLMs** with these models is more nuanced and involves greater care. These are best avoided unless you are familiar with and understand the **utility and pitfalls** of these methods.

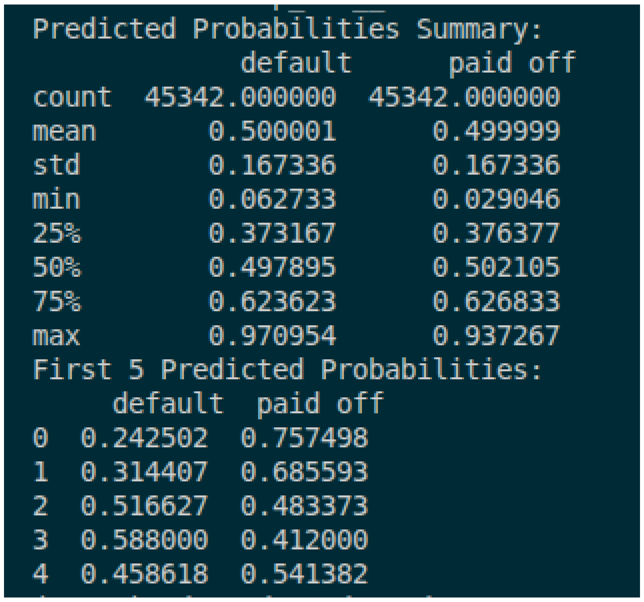
III.4. Predicted Values from Logistic Regression

The predicted value from logistic regression is in terms of the log odds: Y = log(Odds (Y = 1)).

The predicted probability is given by the logistic response function: $\hat{p} = \frac{1}{1 + e^{-\hat{Y}}}$

In Python, we can convert the probabilities into a data frame and use the describe method to get these characteristics of the distribution.

These are on a scale from 0 to 1 and don't yet declare whether the predicted value is default or paid off. We could declare any value greater than 0.5 as default. In practice, a lower cutoff is often appropriate if the goal is to identify members of a rare class (see "The Rare Class Problem" on page 223).



```
Predicted Probabilities Summary:
             default      paid off
count  45342.000000  45342.000000
mean       0.500001      0.499999
std        0.167336      0.167336
min        0.062733      0.029046
25%        0.373167      0.376377
50%        0.497895      0.502105
75%        0.623623      0.626833
max        0.970954      0.937267
First 5 Predicted Probabilities:
    default  paid off
0  0.242502  0.757498
1  0.314407  0.685593
2  0.516627  0.483373
3  0.588000  0.412000
4  0.458618  0.541382
```

interpretation:

- Loans with **predicted default probability close to 1** → high risk → may require closer monitoring or denial.
- Loans with **predicted paid-off probability close to 1** → low risk → likely safe to approve.
- Probabilities around **0.5** → uncertain loans → could trigger manual review or additional checks.

```
    default  paid off
0  0.242502  0.757498
1  0.314407  0.685593
2  0.516627  0.483373
3  0.588000  0.412000
4  0.458618  0.541382
```

- **Row 0:** Loan 0 → 24% chance to default, 76% chance to pay off → model predicts **paid off**.
- **Row 2:** Loan 2 → 52% chance to default, 48% chance to pay off → model predicts **default**.

Rule for prediction: By default, `logit_reg.predict()` chooses the class with **higher probability**.

- If `default` > 0.5 → predict **default**
- If `paid off` > 0.5 → predict **paid off**

Interpreting the Summary Statistics (`describe()`)

```
            default      paid off
count  45342.000000  45342.000000
mean       0.500001      0.499999
std        0.167336      0.167336
min        0.062733      0.029046
25%        0.373167      0.376377
50%        0.497895      0.502105
75%        0.623623      0.626833
max        0.970954      0.937267
```

- **count** → number of loans in the dataset.
- **mean** → average predicted probability for each class (~0.5). This suggests the model predicts roughly half the loans as likely to default and half as likely to pay off.
- **std (standard deviation)** → variability of predicted probabilities. ~0.17 means most predictions are clustered around 0.5 ± 0.17.
- **min / max** → lowest and highest predicted probabilities.
  - e.g., some loans are predicted with only **6% chance to default**, while others with **97% chance**.
- **25%, 50%, 75% (quartiles)** → distribution of predicted probabilities.

III.5. Interpreting the Coefficients and Odds Ratios

One major advantage of **logistic regression** is that once the model is trained, it can be applied to new data **very quickly**—you just plug in the values and get a probability, with no need to recompute anything. Another advantage is **interpretability**: the model explains how each variable affects the outcome in a way that is easy to communicate.

The key idea behind logistic regression is the **odds ratio**.

First, recall what *odds* mean.
If the probability that an event happens is ppp, then the odds are:

$$\text{odds} = \frac{p}{1-p}$$

So odds compare "how likely something happens" to "how likely it does not happen."

Example

Suppose we study loan repayment:

- If **homeowner (X = 1)**:
  - Probability of paying off loan = 0.80
  - Odds = 0.80/0.20=40.80 / 0.20 = 40.80/0.20=4

- If **not homeowner (X = 0)**:
  - Probability of paying off loan = 0.50
  - Odds = 0.50/0.50=10.50 / 0.50 = 10.50/0.50=1

The **odds ratio** is:

$$\frac{4}{1} = 4$$

Interpretation

An odds ratio of **4** means: The odds of paying off the loan are **4 times higher** for homeowners than for non-homeowners.

Why do we bother with odds ratios instead of working directly with probabilities?

The reason is mathematical, but it leads to an important practical benefit. In **logistic regression**, each coefficient βj\beta_jβj is the **logarithm of an odds ratio**. This allows the model to remain linear and easy to estimate, while still being interpretable.
In logistic regression, we do **not** model probabilities directly. Instead, we model the **log-odds** of the outcome as a linear function of the predictors. When we exponentiate a coefficient, we obtain the **odds ratio** associated with that predictor.

Example
Suppose a logistic regression model includes a binary variable called
purpose_small_business, and its estimated coefficient is:
$\beta = 1.21526$
This coefficient is on the **log-odds scale**. To interpret it, we exponentiate it:
$\exp(1.21526) \approx 3.4$

This means:

> Holding all other variables constant, a loan made for a small business has **3.4 times higher odds of default** compared to a loan used to pay off credit card debt.

In other words, loans for creating or expanding a small business are **substantially riskier** than credit card consolidation loans.

Why the log scale matters

Because coefficients are on the log scale, changes in the coefficients translate into **multiplicative changes in the odds**:

- An increase of **1 unit** in a coefficient multiplies the odds by:
  exp(1) ≈ 2.72
- An increase of **2 units** multiplies the odds by:
  exp(2) ≈ 7.4

This shows why even relatively small changes in coefficients can correspond to large changes in risk.

Intuition

- Probabilities are bounded between 0 and 1 and do not combine linearly.
- Log-odds can take any real value, which makes linear modeling possible.
- Odds ratios provide a **clear and interpretable measure of effect size**.

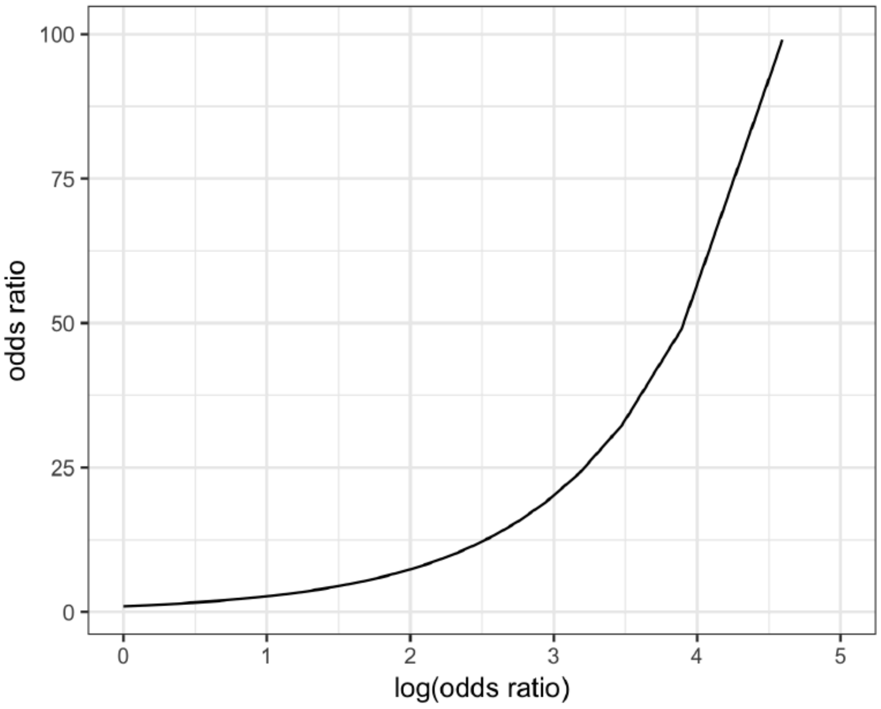


Figure 5-3. The relationship between the odds ratio and the log-odds ratio

For **numeric (continuous) variables**, odds ratios are interpreted in much the same way as for binary variables, but with one key difference: they describe how the **odds change for a one-unit increase in the variable**.

Take the **payment-to-income ratio** as an example. Suppose the estimated coefficient for this variable is 0.08244. Exponentiating it gives:

exp(0.08244) ≈ 1.09

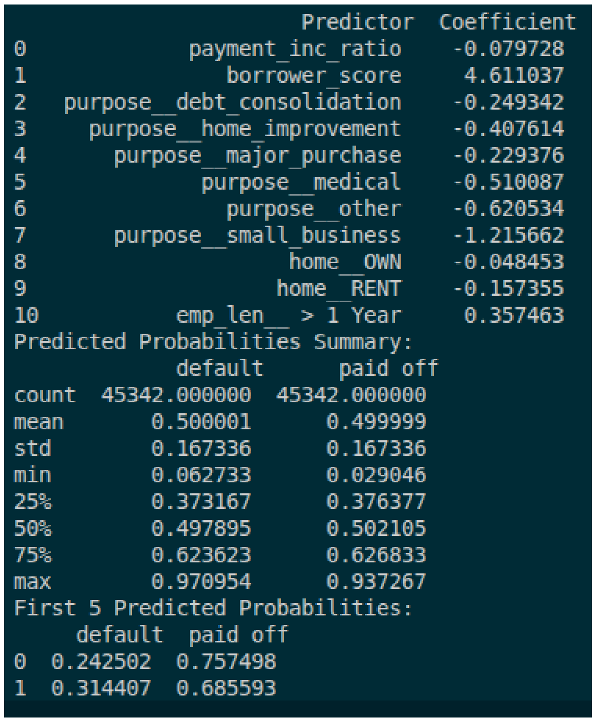
This means that increasing the payment-to-income ratio from 5 to 6 (a one-unit increase) raises the odds of loan default by about **9%**. Each additional unit increases risk multiplicatively, not additively.

Now consider **borrower_score**, which measures creditworthiness and ranges from 0 (worst) to 1 (best). If its coefficient is −4.61264, then:

exp(−4.61264) ≈ 0.01

This tells us that the odds of default for the **best borrowers** are only about **1% of the odds** for the worst borrowers. Put differently, borrowers with the poorest creditworthiness have a default risk that is roughly **100 times higher** than that of the most creditworthy borrowers.

The key idea is that for numeric variables, odds ratios capture how risk **scales multiplicatively** as the variable increases, often revealing large practical effects even when the coefficients themselves seem modest.



```
                       Predictor  Coefficient
0               payment_inc_ratio    -0.079728
1                  borrower_score     4.611037
2    purpose__debt_consolidation    -0.249342
3      purpose__home_improvement    -0.407614
4        purpose__major_purchase    -0.229376
5               purpose__medical    -0.510087
6                 purpose__other    -0.620534
7        purpose__small_business    -1.215662
8                      home__OWN    -0.048453
9                     home__RENT    -0.157355
10             emp_len__ > 1 Year     0.357463
Predicted Probabilities Summary:
             default      paid off
count  45342.000000  45342.000000
mean       0.500001      0.499999
std        0.167336      0.167336
min        0.062733      0.029046
25%        0.373167      0.376377
50%        0.497895      0.502105
75%        0.623623      0.626833
max        0.970954      0.937267
First 5 Predicted Probabilities:
    default  paid off
0  0.242502  0.757498
1  0.314407  0.685593
```

Holding all other variables constant, increasing `borrower_score` by **1 unit** multiplies the **odds of being paid off rather than defaulting** by **about 100**.

β=4.611

To interpret it, we exponentiate exp(4.611)≈100

III.6. Linear and Logistic Regression: Similarities and Differences

What they have in common

Both **linear regression** and **logistic regression** assume a **linear relationship between predictors and the response**—but in different spaces.

- In **linear regression**, predictors combine linearly to predict a numeric outcome:

    - Example: predicting **house price** from size and location.

- In **logistic regression**, predictors combine linearly to predict the **log-odds** of an event:

    - Example: predicting whether a loan **defaults or is paid off**.

In both cases:
- We choose predictors the same way
- We can use interactions, transformations, and splines
- Model selection, diagnostics, and interpretation follow similar logic

So conceptually, logistic regression is a **natural extension of linear regression** to classification problems.

Key difference 1: How the model is fit
Linear regression is fit using **least squares**, which minimizes the sum of squared errors.
This works because the outcome is continuous and normally distributed.
Logistic regression cannot use least squares because:
- The outcome is **binary** (0/1)
- Predictions must stay between **0 and 1**

Instead, logistic regression uses **maximum likelihood estimation**:
- It finds the coefficients that make the observed 0s and 1s most probable
- This ensures valid probabilities

Key difference 2: Residuals behave differently
In linear regression:
- Residuals = observed − predicted
- Residuals are continuous and roughly normal
- We analyze them with residual plots, QQ-plots, etc.

In logistic regression:
- Residuals are not normally distributed
- Each observation is either 0 or 1
- Specialized residuals are used (deviance, Pearson residuals)

Fitting the model

Fitting a model depends on the type of regression. For **linear regression**, we use **least squares**, which finds the line that minimizes the sum of squared differences between predicted and actual values. We check how well it fits using metrics like **RMSE** (measuring prediction error) and **R-squared** (measuring explained variance). For example, predicting house prices from size uses linear regression.

Logistic regression is different because the outcome is categorical (0 or 1). We can't solve it with a simple formula, so we use **maximum likelihood estimation (MLE)**. MLE searches for the model most likely to produce the data we observe. Instead of predicting 0 or 1 directly, logistic regression estimates the **log odds** of the outcome being 1. The MLE adjusts parameters so the predicted log odds match the observed outcomes as closely as possible.

The fitting algorithm is iterative. It uses **quasi-Newton optimization**, alternating between calculating how good the current parameters are (**Fisher scoring**) and updating them to improve the fit. Think of it like climbing a hill in the fog: each step uses the slope where you are to guide you closer to the top—the best-fitting model.

Fortunately, most practitioners don't need to concern themselves with the details of the fitting algorithm since this is handled by the software. Most data scientists will not need to worry about the fitting method, other than understanding that it is a way to find a good model under certain assumptions.

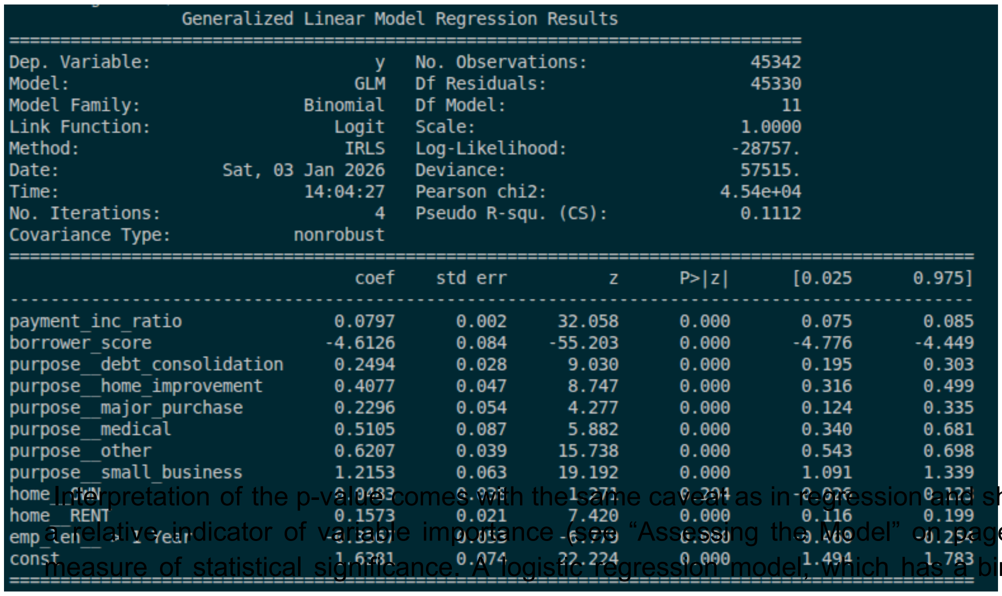
Assessing the model

Like other classification methods, **logistic regression** is judged by how well it **classifies new data**—for example, predicting whether a patient has a disease (1) or not (0). Beyond classification accuracy, we can use standard statistical tools to understand and improve the model.

R reports the **estimated coefficients**, which show the effect of each predictor on the log odds. Along with these, it gives the **standard error (SE)**, which tells us how precise each estimate is; the **z-value**, which measures how many SEs the coefficient is from zero; and the **p-value**, which tests whether the effect is statistically significant.

For instance, if a coefficient for blood pressure has a high z-value and a very low p-value, it strongly suggests blood pressure influences the odds of disease. These metrics help us **interpret the model** and decide which predictors matter most.

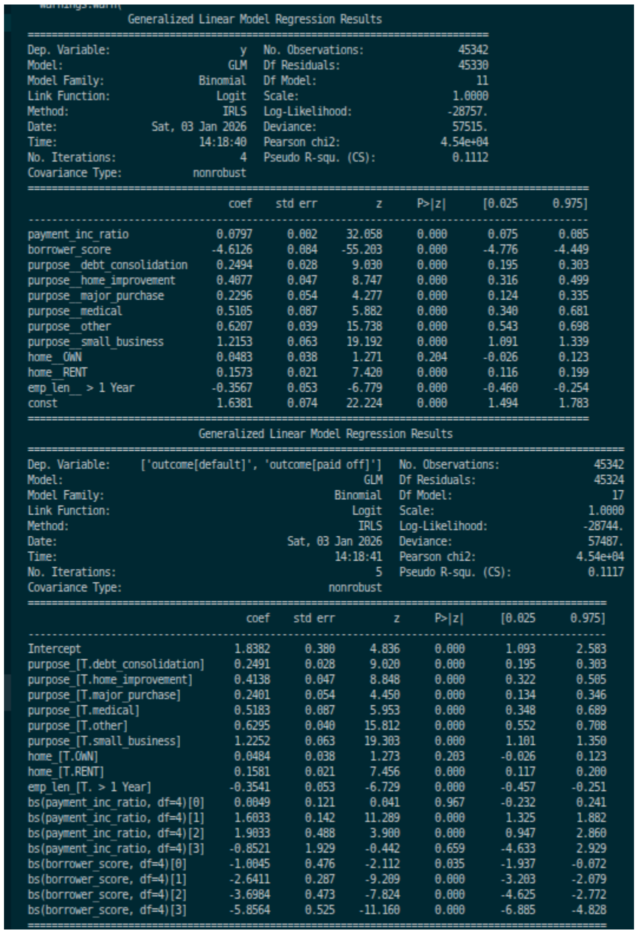
The package statsmodels has an implementation for generalized linear model (GLM) that provides similarly detailed information:



```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                45342
Model:                            GLM   Df Residuals:                    45330
Model Family:                Binomial   Df Model:                           11
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -28757.
Date:                Sat, 03 Jan 2026   Deviance:                       57515.
Time:                        14:04:27   Pearson chi2:                 4.54e+04
No. Iterations:                     4   Pseudo R-squ. (CS):             0.1112
Covariance Type:            nonrobust
=============================================================================================
                                coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------------------
payment_inc_ratio             0.0797      0.002     32.058      0.000       0.075       0.085
borrower_score               -4.6126      0.084    -55.203      0.000      -4.776      -4.449
purpose__debt_consolidation   0.2494      0.028      9.030      0.000       0.195       0.303
purpose__home_improvement     0.4077      0.047      8.747      0.000       0.316       0.499
purpose__major_purchase       0.2296      0.054      4.277      0.000       0.124       0.335
purpose__medical              0.5105      0.087      5.882      0.000       0.340       0.681
purpose__other                0.6207      0.039     15.738      0.000       0.543       0.698
purpose__small_business       1.2153      0.063     19.192      0.000       1.091       1.339
home__OWN                     0.0483      0.038      1.271      0.204      -0.026       0.123
home__RENT                    0.1573      0.021      7.420      0.000       0.116       0.199
emp_len__ > 1 Year           -0.3567      0.053     -6.779      0.000      -0.460      -0.254
const                         1.6381      0.074     22.224      0.000       1.494       1.783
=============================================================================================
```

Interpretation of the p-value comes with the same caveat as in regression and should be viewed more as a relative indicator of variable importance (see "Assessing the Model" on page 153) than as a formal measure of statistical significance. A logistic regression model, which has a binary response, does not have an associated RMSE or R-squared. Instead, a logistic regression model is typically evaluated using more general metrics for classification; see “Evaluating Classification Models” on page 219. Many other concepts for linear regression carry over to the logistic regression setting (and other GLMs). For example, you can use stepwise regression, fit interaction terms, or include spline terms. The same concerns regarding confounding and correlated variables apply to logistic regression (see “Interpreting the Regression Equation” on page 169).

You can fit generalized additive models (see “Generalized Additive Models” on page 192) using statsmodels.



```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                45342
Model:                            GLM   Df Residuals:                    45330
Model Family:                Binomial   Df Model:                           11
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -28757.
Date:                Sat, 03 Jan 2026   Deviance:                       57515.
Time:                        14:18:40   Pearson chi2:                 4.54e+04
No. Iterations:                     4   Pseudo R-squ. (CS):             0.1112
Covariance Type:            nonrobust
===============================================================================================
                                  coef    std err          z      P>|z|      [0.025      0.975]
-----------------------------------------------------------------------------------------------
payment_inc_ratio               0.0797      0.002     32.058      0.000       0.075       0.085
borrower_score                 -4.6126      0.084    -55.203      0.000      -4.776      -4.449
purpose__debt_consolidation     0.2494      0.028      9.030      0.000       0.195       0.303
purpose__home_improvement       0.4077      0.047      8.747      0.000       0.316       0.499
purpose__major_purchase         0.2296      0.054      4.277      0.000       0.124       0.335
purpose__medical                0.5105      0.087      5.882      0.000       0.340       0.681
purpose__other                  0.6207      0.039     15.738      0.000       0.543       0.698
purpose__small_business         1.2153      0.063     19.192      0.000       1.091       1.339
home__OWN                       0.0483      0.038      1.271      0.204      -0.026       0.123
home__RENT                      0.1573      0.021      7.420      0.000       0.116       0.199
emp_len__ > 1 Year             -0.3567      0.053     -6.779      0.000      -0.460      -0.254
const                           1.6381      0.074     22.224      0.000       1.494       1.783
===============================================================================================
                              Generalized Linear Model Regression Results
===================================================================================================
Dep. Variable:     ['outcome[default]', 'outcome[paid off]']   No. Observations:             45342
Model:                                                   GLM   Df Residuals:                 45324
Model Family:                                       Binomial   Df Model:                        17
Link Function:                                         Logit   Scale:                       1.0000
Method:                                                 IRLS   Log-Likelihood:             -28744.
Date:                                       Sat, 03 Jan 2026   Deviance:                    57487.
Time:                                               14:18:41   Pearson chi2:              4.54e+04
No. Iterations:                                            5   Pseudo R-squ. (CS):          0.1117
Covariance Type:                                   nonrobust
================================================================================================
                                   coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------------
Intercept                        1.8382      0.380      4.836      0.000       1.093       2.583
purpose_[T.debt_consolidation]   0.2491      0.028      9.020      0.000       0.195       0.303
purpose_[T.home_improvement]     0.4138      0.047      8.848      0.000       0.322       0.505
purpose_[T.major_purchase]       0.2401      0.054      4.450      0.000       0.134       0.346
purpose_[T.medical]              0.5183      0.087      5.953      0.000       0.348       0.689
purpose_[T.other]                0.6295      0.040     15.812      0.000       0.552       0.708
purpose_[T.small_business]       1.2252      0.063     19.303      0.000       1.101       1.350
home_[T.OWN]                     0.0484      0.038      1.273      0.203      -0.026       0.123
home_[T.RENT]                    0.1581      0.021      7.456      0.000       0.117       0.200
emp_len_[T. > 1 Year]           -0.3541      0.053     -6.729      0.000      -0.457      -0.251
bs(payment_inc_ratio, df=4)[0]   0.0049      0.121      0.041      0.967      -0.232       0.241
bs(payment_inc_ratio, df=4)[1]   1.6033      0.142     11.289      0.000       1.325       1.882
bs(payment_inc_ratio, df=4)[2]   1.9033      0.488      3.900      0.000       0.947       2.860
bs(payment_inc_ratio, df=4)[3]  -0.8521      1.929     -0.442      0.659      -4.633       2.929
bs(borrower_score, df=4)[0]     -1.0045      0.476     -2.112      0.035      -1.937      -0.072
bs(borrower_score, df=4)[1]     -2.6411      0.287     -9.209      0.000      -3.203      -2.079
bs(borrower_score, df=4)[2]     -3.6984      0.473     -7.824      0.000      -4.625      -2.772
bs(borrower_score, df=4)[3]     -5.8564      0.525    -11.160      0.000      -6.885      -4.828
================================================================================================
```

Analysis of residuals

One area where logistic regression differs from linear regression is in the analysis of the residuals. There is currently no implementation of partial residuals in any of the major Python packages. We provide Python code to create the partial residual plot in the accompanying source code repository.

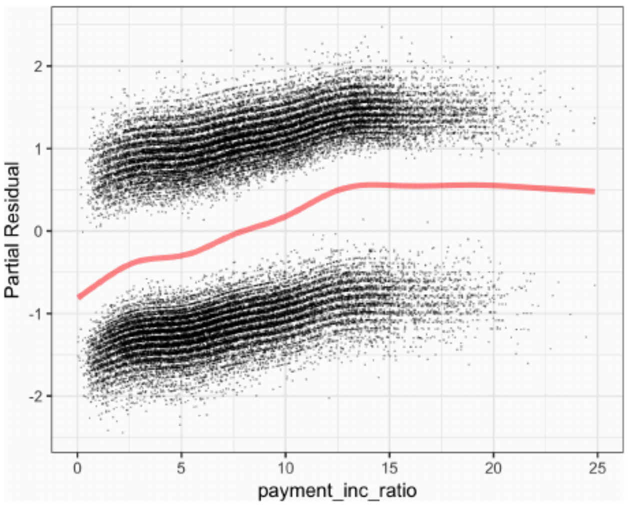


Figure 5-4. Partial residuals from logistic regression

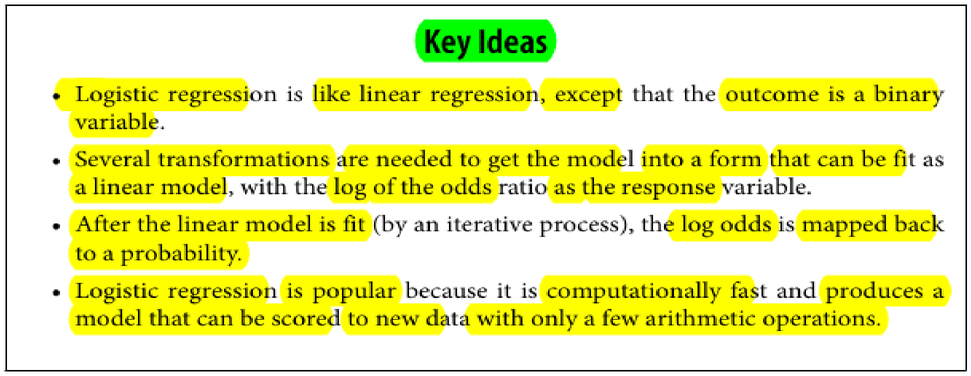
In a **partial residual plot for logistic regression**, you often see two "clouds" of points: one for responses of 1 (e.g., loans that defaulted) and one for 0 (loans paid off). This happens because the **actual outcome is binary**, but the model predicts the **logit**—a continuous number representing the log odds of the event.

Since the logit is always finite, but the true 0/1 corresponds to an "infinite" logit in theory, the residuals **can't be zero**, so points naturally spread above and below the fitted line.

Even though partial residuals in logistic regression are **less precise than in linear regression**, they are still useful:

- To **check for nonlinear relationships**, e.g., seeing if the effect of payment-to-income ratio bends rather than being straight.
- To **spot highly influential points**, such as extreme loan values that strongly affect the model fit.

Example: In your plot, the red line shows the estimated effect of `payment_inc_ratio` on the log odds of default. The two point clouds show the actual loans: top for defaults, bottom for paid-off loans. You can see how the predicted curve fits **between these clouds**, capturing the general trend while highlighting outliers.



Key Ideas

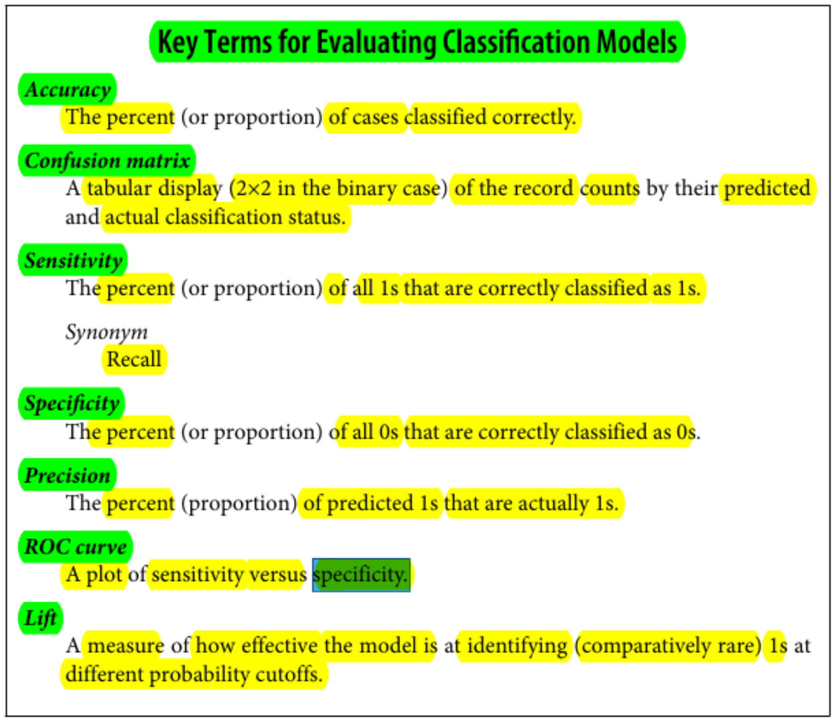
- Logistic regression is like linear regression, except that the outcome is a binary variable.
- Several transformations are needed to get the model into a form that can be fit as a linear model, with the log of the odds ratio as the response variable.
- After the linear model is fit (by an iterative process), the log odds is mapped back to a probability.
- Logistic regression is popular because it is computationally fast and produces a model that can be scored to new data with only a few arithmetic operations.

IV. Evaluating Classification Models

In **predictive modeling**, it's common to train several different models and compare how well they perform on **data they haven't seen before**, called a **holdout sample**. This helps ensure the model isn't just memorizing the training data but can generalize to new cases.

Often, data are split into **training** and **validation (or test)** sets. After testing and tuning multiple models—say, a logistic regression, a decision tree, and a random forest—the best-performing one is selected. If enough data are available, a **third holdout set** may be used at the very end to estimate how the chosen model will perform on **completely new data**.

For example, when predicting loan default, you might train models on past loans, validate them on a separate subset, and finally test the chosen model on a fresh set of recent loans. Different fields may use terms like *validation* or *test* differently, but the goal is always the same: **identify the model that gives the most accurate and useful predictions in practice**.



Key Terms for Evaluating Classification Models

Accuracy
The percent (or proportion) of cases classified correctly.

Confusion matrix
A tabular display (2×2 in the binary case) of the record counts by their predicted and actual classification status.

Sensitivity
The percent (or proportion) of all 1s that are correctly classified as 1s.

Synonym
Recall

Specificity
The percent (or proportion) of all 0s that are correctly classified as 0s.

Precision
The percent (proportion) of predicted 1s that are actually 1s.

ROC curve
A plot of sensitivity versus specificity.

Lift
A measure of how effective the model is at identifying (comparatively rare) 1s at different probability cutoffs.

A simple way to measure classification performance is to count the proportion of pre- dictions that are correct, i.e., measure the accuracy. Accuracy is simply a measure of total error:

$$\text{accuracy} = \frac{\sum \text{TruePositive} + \sum \text{TrueNegative}}{\text{SampleSize}}$$

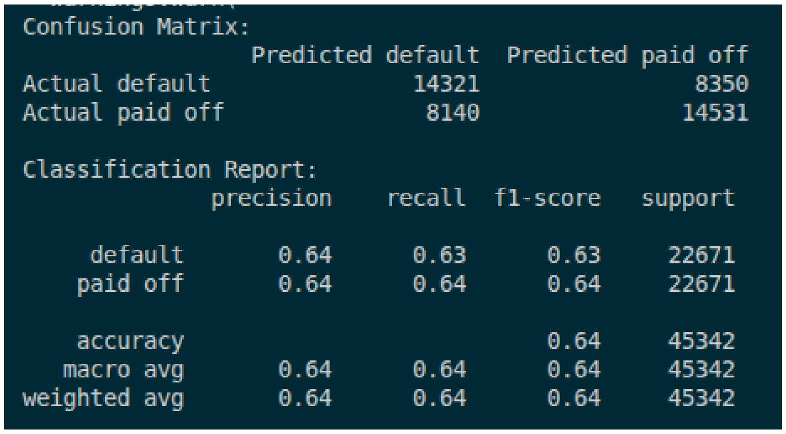
In most classification algorithms, each case is assigned an "estimated probability of being a 1."3 The default decision point, or cutoff, is typically 0.50 or 50%. If the probability is above 0.5, the classification is "1"; otherwise it is "0." An alternative default cutoff is the prevalent probability of 1s in the data.

IV.1. Confusion Matrix

At the core of **classification evaluation** is the **confusion matrix**, a simple table that summarizes how well a model's predictions match the actual outcomes. It counts **correct and incorrect predictions**, broken down by type of response. While R and Python offer built-in tools to compute it, in binary classification it's easy to understand and even calculate by hand.

Consider a **logistic GAM model** trained on a **balanced dataset**, where half the loans defaulted and half were paid off. By convention, **Y = 1** represents the event of interest (loan default), and **Y = 0** represents the normal outcome (loan paid off). The confusion matrix then shows how many defaults were correctly predicted as defaults, how many paid-off loans were correctly identified, and where the model made mistakes by confusing one for the other.

This table forms the basis for many key metrics—such as accuracy, precision, and recall—and helps us understand not just how often the model is right, but **how it is wrong**, which is often more important in practice.

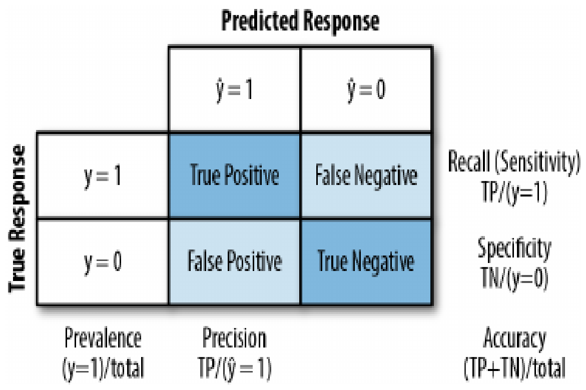


```
Confusion Matrix:
                 Predicted default  Predicted paid off
Actual default               14321                8350
Actual paid off               8140               14531

Classification Report:
              precision    recall  f1-score   support

     default       0.64      0.63      0.63     22671
    paid off       0.64      0.64      0.64     22671

    accuracy                           0.64     45342
   macro avg       0.64      0.64      0.64     45342
weighted avg       0.64      0.64      0.64     45342
```



The predicted outcomes are columns and the true outcomes are the rows. The diagonal elements of the matrix show the number of correct predictions, and the off diagonal elements show the number of incorrect predictions. For example, 14,295 defaulted loans were correctly predicted as a default, but 8,376 defaulted loans were incorrectly predicted as paid off.

The **diagonal cells** are the moments when the model gets it right.

- **14,321 true positives**: loans that actually defaulted and were predicted to default.
- **14,531 true negatives**: loans that were paid off and correctly predicted as such.

The **off-diagonal cells** show mistakes.

- **8,350 false negatives**: defaults that the model missed (predicted "paid off").
- **8,140 false positives**: loans predicted to default but that actually paid off.

From this table, all the familiar metrics emerge naturally. **Recall (sensitivity)** focuses on how many actual defaults were caught: the model finds many defaults, but it still misses a substantial number. **Precision**, on the other hand, asks: *when the model predicts "default," how often is it right?* Here, false positives are large, so precision is weakened.

An especially important idea is the **false positive rate**—how often the model cries "default" when there isn't one. When the event of interest (Y = 1) is **rare**, even a good model can generate many false alarms. This creates a counterintuitive outcome: **most predicted positives may actually be negatives**.

This is exactly the issue seen in **medical screening**. For example, breast cancer is relatively rare in the general population. Even with a highly accurate mammogram, most *positive* test results end up being false positives. People often assume a positive result means "I almost certainly have the disease," but statistically, it usually does not.

Interpretation of classification report:

This **classification report** is simply a compact summary of what the confusion matrix is already telling you, but translated into **performance metrics** that are easier to compare across models.
Here's how to read it—**as a narrative**.
The model treats **"default"** and **"paid off"** symmetrically. Both classes have **about the same number of observations** (22,671 each), so this is a **balanced dataset**. That's important: none of the metrics are being artificially inflated by class imbalance.
For **default** loans:

- **Precision = 0.64**
  When the model predicts *default*, it is correct about **64% of the time**. Roughly 1 out of 3 predicted defaults is actually a paid-off loan.

- **Recall = 0.63**
  The model catches about **63% of all actual defaults**, but misses the remaining 37%.

- **F1-score = 0.63**
  This balances precision and recall, indicating **moderate but not strong** default detection.

For **paid-off** loans:

- Precision, recall, and F1 are all **~0.64**, meaning the model is **equally good (and equally imperfect)** at identifying non-defaults.

The **overall accuracy = 0.64** means:
The model makes the correct classification in about **64% of cases**.
Because the data are balanced, accuracy is meaningful here. In an imbalanced setting, this number could be misleading.
The **macro average** simply averages the metrics across the two classes, treating them equally.
The **weighted average** does the same but weights by class size—here they are identical because the classes are equal in size.

IV.2. The Rare Class Problem

In many real-world classification problems, **the classes are not balanced**. One outcome is rare but important—fraudulent insurance claims, loan defaults, medical conditions, or customers who actually make a purchase. This rare outcome is usually labeled **1**, while the common, routine case is labeled **0**.

What matters most in these situations is **not overall accuracy**, but **what kind of mistakes the model makes**.

Take insurance fraud as an example. Missing a fraudulent claim (predicting 0 when the truth is 1) can cost **thousands of dollars**, while flagging a legitimate claim as suspicious only adds some extra review time. These two errors are **not equally costly**, so treating them as if they were is misleading.

This is why accuracy can be dangerously deceptive. Imagine an online store where only **0.1% of visitors make a purchase**. A model that predicts *“no purchase” for everyone* will be **99.9% accurate**—yet completely useless. It finds none of the buyers, which are the only people we actually care about.

In practice, we prefer a model that may be **less accurate overall**, but that is **good at finding the rare, valuable cases**. Such a model might incorrectly flag some nonbuyers or nonfraud cases, but as long as it captures a meaningful fraction of the true positives, it delivers real value.

This is why, in imbalanced problems, metrics like **recall(sensitivity), precision, false positive rate, ROC curves, and cost-based evaluation** matter far more than accuracy. The goal is not to be “mostly right,” but to be right **where it counts**.

IV.3. Precision, Recall, and Specificity

Metrics other than pure accuracy—metrics that are more nuanced—are commonly used in evaluating classification models.

Several of these have a long history in statistics—especially biostatistics, where they are used to describe the expected performance of diagnostic tests. The precision measures the accuracy of a predicted positive outcome (see Figure 5-5):

$$\text{precision} = \frac{\sum \text{TruePositive}}{\sum \text{TruePositive} + \sum \text{FalsePositive}}$$

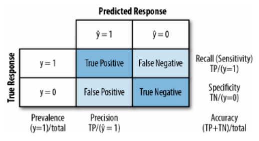


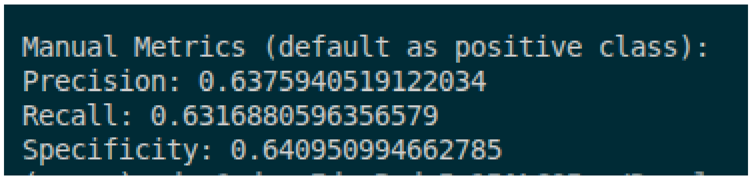
Figure 5-5. Confusion matrix for a binary response and various metrics

The recall, also known as sensitivity, measures the strength of the model to predict a positive outcome—the proportion of the 1s that it correctly identifies (see Figure 5-5). The term sensitivity is used a lot in biostatistics and medical diagnostics, whereas recall is used more in the machine learning community. The definition of recall is:

$$\text{recall} = \frac{\sum \text{TruePositive}}{\sum \text{TruePositive} + \sum \text{FalseNegative}}$$

Another metric used is specificity, which measures a model’s ability to predict a negative outcome:

$$\text{specificity} = \frac{\sum \text{TrueNegative}}{\sum \text{TrueNegative} + \sum \text{FalsePositive}}$$



```
Manual Metrics (default as positive class):
Precision: 0.6375940519122034
Recall: 0.6316880596356579
Specificity: 0.640950994662785
```

IV.4. ROC Curve

You can see that there is a trade-off between recall and specificity. Capturing more 1s generally means misclassifying more 0s as 1s. The ideal classifier would do an excel- lent job of classifying the 1s, without misclassifying more 0s as 1s. The metric that captures this trade-off is the “Receiver Operating Characteristics” curve, usually referred to as the ROC curve. The ROC curve plots recall (sensitivity) on the y-axis against specificity on the x-axis.4 The ROC curve shows the trade-off between recall and specificity as you change the cutoff to determine how to classify a record. Sensitivity (recall) is plotted on the y-axis, and you may encounter two forms in which the x-axis is labeled:

- Specificity plotted on the x-axis, with 1 on the left and 0 on the right
- 1-Specificity plotted on the x-axis, with 0 on the left and 1 on the right

The curve looks identical whichever way it is done. The process to compute the ROC curve is:

1. Sort the records by the predicted probability of being a 1, starting with the most probable and ending with the least probable.
2. Compute the cumulative specificity and recall based on the sorted records.

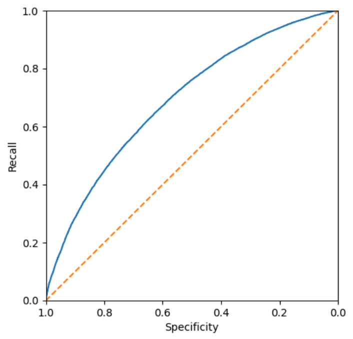


Figure 5-6 illustrates how to interpret an ROC curve. The **dotted diagonal line** represents a classifier that is doing **no better than random guessing**. At any cutoff, it identifies 1s and 0s purely by chance, so recall increases only by sacrificing specificity at the same rate.

A **highly effective classifier**, by contrast, produces an ROC curve that **hugs the upper-left corner** of the plot. This shape means the model is able to correctly identify many 1s (high recall) while still keeping misclassification of 0s low (high specificity). In medical testing, this corresponds to detecting most patients with a disease without falsely alarming too many healthy patients.

In this model, the ROC curve shows a useful trade-off. If we require a **specificity of at least 50%**—meaning that at least half of the 0s are correctly classified—then the model still achieves a **recall of about 75%**. In practical terms, this means the model can catch roughly three-quarters of the true positives while accepting that some negatives will be misclassified. This kind of insight is exactly what the ROC curve is designed to provide: it helps choose a cutoff that balances competing goals rather than relying on a single accuracy number.

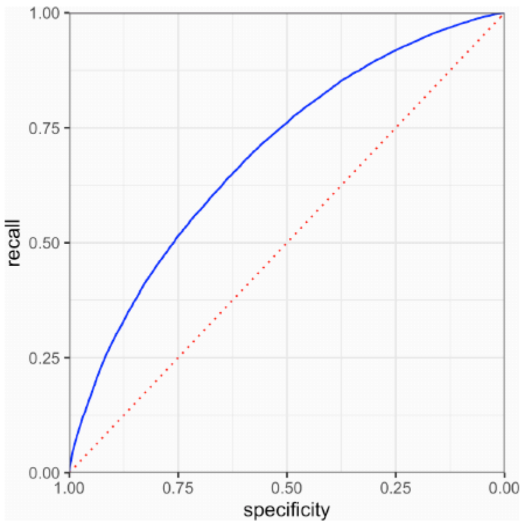


Figure 5-6. ROC curve for the loan data

Figure 5-6 illustrates how to interpret an **ROC curve**. The **dotted diagonal line** represents a classifier that is doing **no better than random guessing**. At any cutoff, it identifies 1s and 0s purely by chance, so **recall increases only by sacrificing specificity at the same rate**.

A **highly effective classifier**, by contrast, produces an ROC curve that **hugs the upper-left corner** of the plot. This shape means the model is able to **correctly identify many 1s (high recall)** while still keeping **misclassification of 0s low (high specificity)**. In medical testing, this corresponds to **detecting most patients with a disease without falsely alarming too many healthy patients**.

In this model, the ROC curve shows a **useful trade-off**. If we require a **specificity of at least 50%**—meaning that **at least half of the 0s are correctly classified**—the model still achieves a **recall of about 75%**. In practical terms, this means the model can **catch roughly three-quarters of the true positives** while accepting that **some negatives will be misclassified**. This is exactly what the ROC curve is designed to show: **how to choose a cutoff that balances competing goals**, rather than relying on a single accuracy number.

Precision-Recall Curve

In addition to ROC curves, it can be illuminating to examine the precision-recall (PR) curve. PR curves are computed in a similar way except that the data is ordered from least to most probable and cumulative precision and recall statistics are computed. PR curves are especially useful in evaluating data with highly unbalanced outcomes.

IV.5. AUC

The ROC curve is a valuable graphical tool, but by itself doesn't constitute a single measure for the performance of a classifier. The ROC curve can be used, however, to produce the area underneath the curve (AUC) metric. AUC is simply the total area under the ROC curve. The larger the value of AUC, the more effective the classifier. An AUC of 1 indicates a perfect classifier: it gets all the 1s correctly classified, and it doesn't misclassify any 0s as 1s.

A completely ineffective classifier—the diagonal line—will have an AUC of 0.5.

Figure 5-7 shows the area under the ROC curve for the loan model. (there is no code in my vscode for this plot).

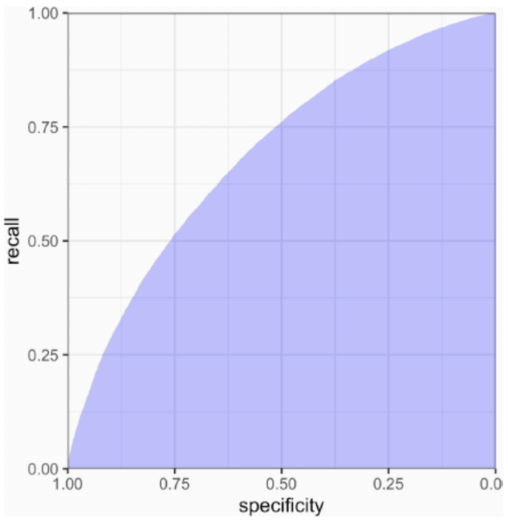
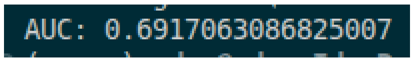


Figure 5-7. Area under the ROC curve for the loan data

In Python, we can either calculate the accuracy as shown for R or use scikit-learn's function sklearn.metrics.roc_auc_score. You will need to provide the expected value as 0 or 1:



```
AUC: 0.6917063086825007
```

False Positive Rate Confusion

False positive and false negative rates are **frequently misunderstood** and are often **mixed up with sensitivity and specificity**, even in academic papers and software documentation. The confusion comes from the fact that similar terms are used to describe **different denominators**.

In the strict statistical sense, the **false positive rate (FPR)** is the **proportion of true negatives that are incorrectly classified as positives**. For example, in loan default prediction, it is the fraction of paid-off loans that are wrongly flagged as defaults. This definition is directly tied to **specificity**, since
FPR = 1 − specificity.

However, in some applied fields—such as **network intrusion detection or alert systems**—the term "false positive rate" is often used differently. There, it may refer to the **proportion of positive alerts that are actually false**, meaning the fraction of alarms that turn out to be normal behavior. Statistically, this is **not** the false positive rate; it is **1 − precision**.

For example, if a security system raises 100 alerts and 80 of them are harmless, practitioners may say the false positive rate is 80%, even though, statistically, that quantity measures **how unreliable a positive signal is**, not how often negatives are misclassified.

Because of this ambiguity, it is important to **always check the definition being used**, especially when comparing models or reading results across disciplines. The same term can describe **very different error behaviors**, depending on context.

IV.6. Lift

Using the AUC as a metric to evaluate a model is an improvement over simple accu- racy, as it can assess how well a classifier handles the trade-off between overall accuracy and the need to identify the more important 1s. But it does not completely address the rare-case problem, where you need to lower the model's probability cutoff below 0.5 to avoid having all records classified as 0. In such cases, for a record to be classified as a 1, it might be sufficient to have a probability of 0.4, 0.3, or lower. In effect, we end up overidentifying 1s, reflecting their greater importance.

Changing this cutoff will improve your chances of catching the 1s (at the cost of misclassifying more 0s as 1s). But what is the optimum cutoff?

The concept of **lift** lets you sidestep this question for a moment. Instead of choosing a cutoff immediately, you **rank the records by their predicted probability of being 1**. Then you ask: *how much better is the model at identifying 1s than random selection?*

For example, suppose the **top 10% of records**, ranked by predicted probability, contains **0.3% actual 1s**, while the overall dataset has **0.1% 1s**. This means the model is **3 times better than random** in this top decile. That ratio is called the **lift** (or gains) in that segment.

A **lift chart (or gains chart)** visualizes this across the dataset. You can compute lift **decile by decile**, or continuously, to see how much better the model performs than blind selection at each probability level.

Scenario

- We have **10,000 loan applications**.
- Only **100 are defaults** (so 1% overall default rate).

Our model predicts a probability of default for each loan.

Step 1: Rank by predicted probability

We **sort the loans** from highest predicted probability to lowest.

Let's focus on the **top 10%** of loans, i.e., the **1,000 loans with highest predicted default probability**.

Step 2: Count actual defaults in top decile

- Among these 1,000 loans, **30 actually default**.
- The overall default rate in the dataset is **100 / 10,000 = 0.01 (1%)**.
- In the top 10%, the default rate is **30 / 1,000 = 0.03 (3%)**.

Step 3: Compute lift

$$\text{Lift} = \frac{\text{Default rate in top decile}}{\text{Overall default rate}} = \frac{0.03}{0.01} = 3$$

✅ Interpretation: **The model is 3 times better than random at finding defaults** in the top 10% of predictions.

Step 4: Decile by decile

If we do this for each 10% segment of the data, we get a **lift chart**:

cile (top to bottc	Defaults	Default rate	Lift
Top 10%	30	3%	3
10–20%	20	2%	2
20–30%	15	1.50%	1.5
30–100%	35	0.58%	0.58

The **top deciles** show the **highest lift**, meaning these are the loans most likely to default.

- Lower deciles may have **lift < 1**, meaning the model performs worse than random in that segment.

To build a lift chart, you usually start with a cumulative gains chart.

- On the **y-axis**, you plot **recall** (the proportion of actual 1s captured).
- On the **x-axis**, you plot the **total number of records** (usually ranked by predicted probability of being 1).

For example, imagine we have 10,000 loan applications and 100 defaults. We rank the loans from **most likely to default to least likely**.

- If we look at the **top 1,000 loans (top 10%)**, and find **30 defaults**, the cumulative recall is **30 / 100 = 30%**.
- Continuing down the list, the top 2,000 loans may include **50 defaults**, so cumulative recall is **50 / 100 = 50%**.

The **lift curve** is then created by comparing this cumulative recall to the **diagonal line of random selection**:

- Random selection would capture **10% of defaults in the top 10% of loans**, **20% in the top 20%**, and so on.

- In our example, the model captured **30% in the top 10%**, compared to 10% for random → **lift = 3** in that decile.

Decile gains charts break the dataset into **10% segments** and compute lift for each segment.

Uplift

The term **uplift** is sometimes used interchangeably with **lift**, but it also has a **more specific meaning** in certain contexts.

In particular, uplift is used when a **treatment or intervention** (like A vs. B in an A/B test) is included as a predictor in a model. The goal is to measure the **additional effect of one treatment over the other** on an individual case.

- To compute uplift for a person, you **score the model twice**: once assuming they received **treatment A**, and once assuming **treatment B**.
- The **difference in predicted response** between these two scores is the **uplift** for that individual.

Example:

- A company runs an A/B marketing test: Email A vs. Email B.
- A predictive model estimates the probability a customer will make a purchase if they receive Email A versus Email B.
- For one customer, the model predicts a **20% chance of purchase with A** and **12% with B**.
- The **uplift** for this customer is **20% − 12% = 8%**, meaning Email A is predicted to be more effective for them.

Marketers and political campaigns use this approach to **target individuals with the treatment most likely to influence them**, rather than applying the same treatment to everyone.

In short:

- **Lift**: measures improvement over random selection.
- **Uplift**: measures improvement due to a specific treatment versus another treatment, **at the individual level**.

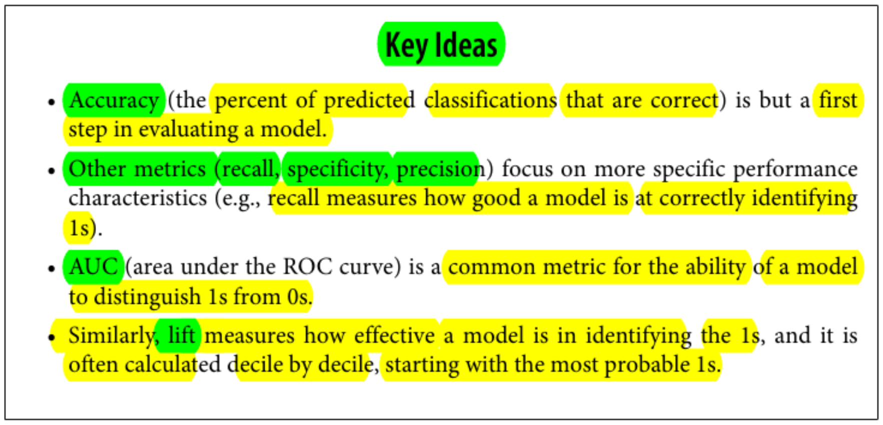
A **lift curve** helps you understand the impact of choosing **different probability cutoffs** for classifying records as 1s (positives). It doesn't give you a single "best" cutoff but shows **how the model performs at different thresholds**, making it easier to decide where to act.

Example:

- A tax authority wants to audit **likely tax cheats** but has limited staff and resources.
- Using a lift chart, it can rank tax returns by the model's predicted probability of fraud.

- The chart shows how many actual tax cheats are captured in the **top-ranked returns**.
- Based on available resources, the authority might decide to audit the **top 5% of returns**, knowing this segment contains the **highest concentration of likely frauds**.

In essence, the lift chart allows decision-makers to **balance effort and impact**, focusing resources on the cases most likely to yield results, rather than acting blindly or using a fixed cutoff.

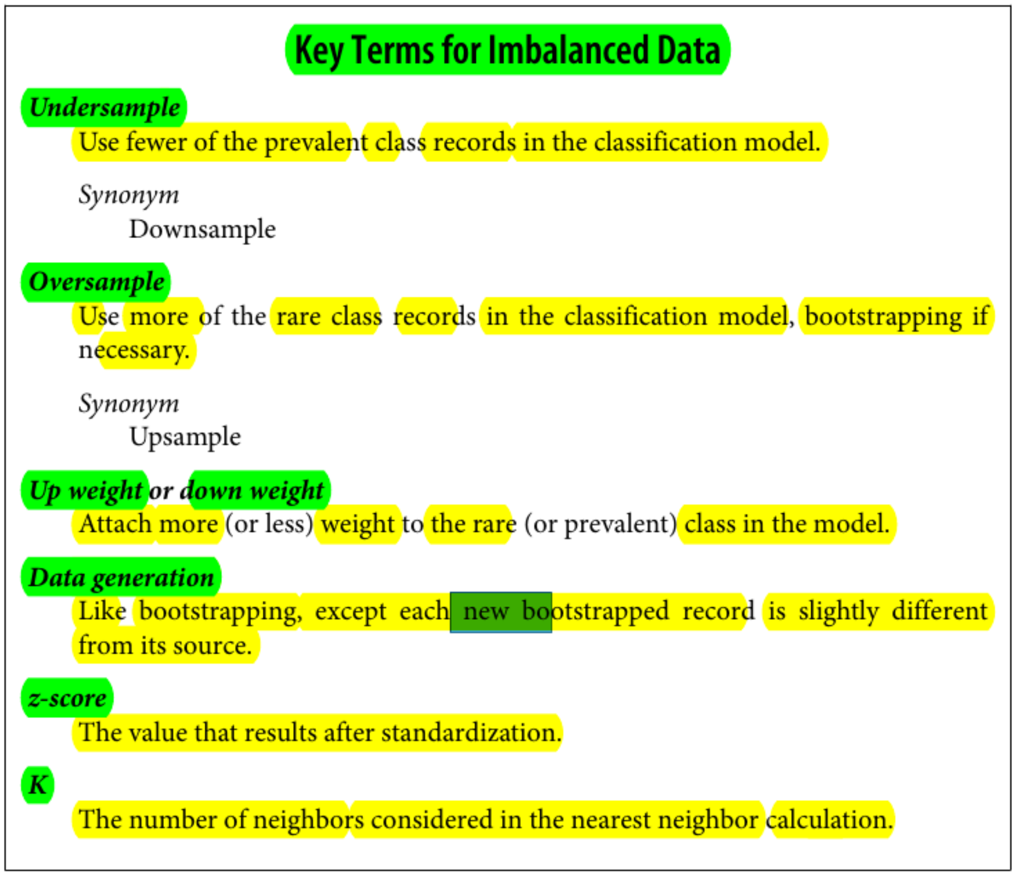


Key Ideas

- Accuracy (the percent of predicted classifications that are correct) is but a first step in evaluating a model.
- Other metrics (recall, specificity, precision) focus on more specific performance characteristics (e.g., recall measures how good a model is at correctly identifying 1s).
- AUC (area under the ROC curve) is a common metric for the ability of a model to distinguish 1s from 0s.
- Similarly, lift measures how effective a model is in identifying the 1s, and it is often calculated decile by decile, starting with the most probable 1s.

V. Strategies for Imbalanced Data

The previous section dealt with evaluation of classification models using metrics that go beyond simple accuracy and are suitable for imbalanced data—data in which the outcome of interest (purchase on a website, insurance fraud, etc.) is rare. In this section, we look at additional strategies that can improve predictive modeling performance with imbalanced data.



Key Terms for Imbalanced Data

Undersample
Use fewer of the prevalent class records in the classification model.

Synonym
Downsample

Oversample
Use more of the rare class records in the classification model, bootstrapping if necessary.

Synonym
Upsample

Up weight or down weight
Attach more (or less) weight to the rare (or prevalent) class in the model.

Data generation
Like bootstrapping, except each new bootstrapped record is slightly different from its source.

z-score
The value that results after standardization.

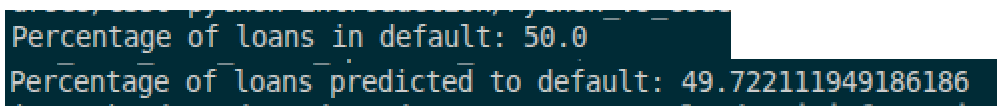
K
The number of neighbors considered in the nearest neighbor calculation.

V.1. Undersampling

If you have enough data, as is the case with the loan data, one solution is to undersample (or downsample) the prevalent class, so the data to be modeled is more balanced between 0s and 1s. The basic idea in undersampling is that the data for the dominant class has many redundant records. Dealing with a smaller, more balanced data set yields benefits in model performance, and it makes it easier to prepare the data and to explore and pilot models.

How much data is enough? It depends on the application, but in general, having tens of thousands of records for the less dominant class is enough. The more easily distinguishable the 1s are from the 0s, the less data needed.

The loan data analyzed in "Logistic Regression" on page 208 was based on a balanced training set: half of the loans were paid off, and the other half were in default. The predicted values were similar: half of the probabilities were less than 0.5, and half were greater than 0.5.



```
Percentage of loans in default: 50.0
Percentage of loans predicted to default: 49.722111949186186
```

The book has another result different that what we found above for predicted to default: 0.39%.

1️⃣ **Imbalanced dataset → 0.39% predicted defaults**

- In the real dataset, **almost all loans are paid off**, and only a tiny fraction actually default (~0.39%).

- Logistic regression is trained on **all the data equally**, so it tries to **minimize overall errors**.

Intuition example:
- Imagine 10,000 loans, only 40 actually default.
- If the model predicts *paid off* for most loans, it is "right" for 9,960 of them.
- Even a real default loan might look similar to paid-off loans due to chance in predictor values.
- Result: **only ~0.39% of loans are predicted to default**, much fewer than the actual number.

✅ Key idea: **The majority class overwhelms the rare class.**

2️⃣ **Balanced dataset → ~50% predicted defaults**
- If we take a **balanced sample** with roughly equal numbers of defaults and paid-off loans:
- The model sees defaults as "normal," not rare.
- It predicts defaults about **half the time**.

Intuition example:
- Imagine 100 loans: 50 defaults, 50 paid off.
- Logistic regression treats both classes equally → predicts **~50% defaults**.

✅ Key idea: **Balancing the dataset helps detect defaults but changes probabilities from reality.**

3️⃣ **Why this matters**
- **Imbalanced dataset** → realistic probabilities, poor detection of rare events.

- **Balanced dataset** → good detection of rare events, probabilities no longer match reality.
- **Lesson:** Accuracy alone is misleading; we need **ROC curves, lift charts, or weighted models** to properly detect rare events like defaults.

V.2. Oversampling and Up/Down Weighting

Undersampling balances a dataset by removing records from the majority class, but this comes at the cost of losing valuable information, especially in small datasets where the rare class already has only a few hundred or thousand records. For example, if you have 10,000 paid-off loans and 200 defaults, removing most paid-off loans to match the 200 defaults throws away patterns that help the model recognize safe borrowers. A better approach in this case is **oversampling** the minority class—duplicating or bootstrapping the rare records—so the dataset becomes balanced without discarding majority-class data, allowing the model to learn from all available information while still giving the rare cases enough weight.

You can achieve a similar effect by weighting the data. Many classification algorithms take a weight argument that will allow you to up/down weight the data.

Most scikit-learn methods allow specifying weights in the fit function using the keyword argument sample_weight.

The weights for loans that default are set to 1\p, where p is the probability of default. The nondefaulting loans have a weight of 1. The sums of the weights for the defaulting loans and nondefaulting loans are roughly equal. The mean of the predicted values is now about 58% instead of 0.39%. Note that weighting provides an alternative to both upsampling the rarer class and downsampling the dominant class.

Adapting the Loss Function

Many classification and regression algorithms optimize a certain criteria or loss function. For example, logistic regression attempts to minimize deviance. In the literature, some propose to modify the loss function in order to avoid the problems caused by a rare class. In practice, this is hard to do: classification algorithms can be complex and difficult to modify. Weighting is an easy way to change the loss function, discounting errors for records with low weights in favor of records with higher weights.

V.3. Data Generating

A variation of upsampling via bootstrapping (see “Oversampling and Up/Down Weighting” on page 232) is data generation by perturbing existing records to create new records. The intuition behind this idea is that since we observe only a limited set of instances, the algorithm doesn’t have a rich set of information to build classification “rules.” By creating new records that are similar but not identical to existing records, the algorithm has a chance to learn a more robust set of rules. This notion is similar in spirit to ensemble statistical models such as boosting and bagging (see Chapter 6).

The idea gained traction with the publication of the SMOTE algorithm, which stands for “Synthetic Minority Oversampling Technique.” The SMOTE algorithm finds a record that is similar to the record being upsampled (see “K-Nearest Neighbors” on page 238) and creates a synthetic record that is a randomly weighted average of the original record and the neighboring record, where the weight is generated separately for each predictor. The number of synthetic oversampled records created depends on the oversampling ratio required to bring the data set into approximate balance with respect to outcome classes.

The SMOTE algorithm, or **Synthetic Minority Oversampling Technique**, is a clever way to handle imbalanced data. Instead of just duplicating rare records, SMOTE creates **new synthetic instances**. It does this by finding a similar record (a “neighbor”) and generating a new record that is a **weighted mix** of the two, with the weights chosen randomly for each predictor.

Example: Suppose your loan dataset has 1,000 paid-off loans and only 10 defaults. To balance the classes, SMOTE might take one default record, find a similar default neighbor, and generate a new synthetic default record somewhere in between these two. Repeating this process, you can create enough synthetic defaults so that the dataset has roughly equal numbers of defaults and paid-offs. This helps models learn patterns for the rare class without just copying existing rows.

The Python package imbalanced-learn implements a variety of methods with an API that is compatible with scikit-learn. It provides various methods for over- and undersampling and support for using these techniques with boosting and bagging classifiers.

V.4. Cost-Based Classification

In practice, simply using **accuracy** or **AUC** to choose a classification cutoff is often too simplistic. A smarter approach is to consider the **costs of mistakes**. For instance, in a loan scenario:

- Let **C** be the expected loss if a loan defaults, and **R** the expected return if it is paid off.
- The **expected return** for a loan is then:

expected return=P(Y=0)×R+P(Y=1)×C\text{expected return} = P(Y=0) \times R + P(Y=1) \times Cexpected return=P(Y=0)×R+P(Y=1)×C

Here, P(Y=1)P(Y=1)P(Y=1) is the predicted probability of default.

Instead of labeling loans simply as “default” or “paid off,” we can use the expected return to make decisions. A loan with a slightly higher default probability but **higher total value** might be more profitable than a smaller, safer loan.

Example:

- Loan A: $1,000, 5% default probability → expected return ≈ $950

  Expected return=0.95×1000+0.05×0=950

- Loan B: $10,000, 8% default probability → expected return ≈ $9,200

  Expected return=0.92×10,000+0.05×0= 9200

Even though Loan B is riskier, it's more profitable. Predicted probabilities are just an **intermediate step**; the real decision metric is **expected profit**, which guides business strategy.

V.5. Exploring the Predictions

A single metric, such as AUC, cannot evaluate all aspects of the suitability of a model for a situation. Figure 5-8 displays the decision rules for four different models fit to the loan data using just two predictor variables: borrower_score and payment_inc_ratio. The models are:

- **Linear Discriminant Analysis (LDA):** Produces a linear decision boundary; in this case, gives nearly identical results to logistic linear regression.
- **Logistic Linear Regression:** Also produces a linear boundary, similar to LDA.
- **Logistic Regression fit using a Generalized Additive Model (GAM):** Produces a smooth, flexible boundary that is a compromise between the linear models and the tree model.
- **Tree Model:** Produces the least regular rule, with step-like regions for classification (see "Tree Models" on page 249).

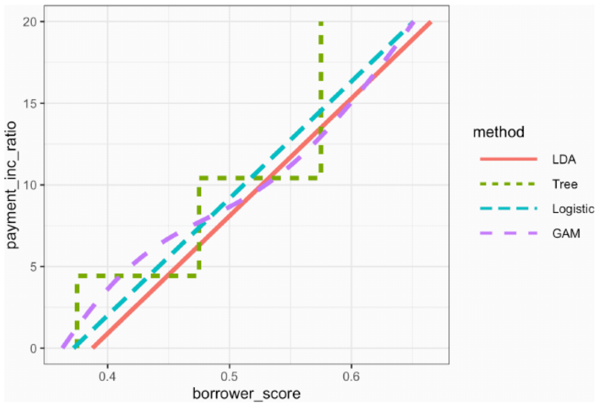


Figure 5-8. Comparison of the classification rules for four different methods

It is not easy to visualize the prediction rules in higher dimensions or, in the case of the GAM and the tree model, even to generate the regions for such rules.

In any case, exploratory analysis of predicted values is always warranted.

Key Ideas

- Highly imbalanced data (i.e., where the interesting outcomes, the 1s, are rare) are problematic for classification algorithms.
- One strategy for working with imbalanced data is to balance the training data via undersampling the abundant case (or oversampling the rare case).
- If using all the 1s still leaves you with too few 1s, you can bootstrap the rare cases, or use SMOTE to create synthetic data similar to existing rare cases.
- Imbalanced data usually indicates that correctly classifying one class (the 1s) has higher value, and that value ratio should be built into the assessment metric.

Summary

Classification, the process of predicting which of two or more categories a record belongs to, is a fundamental tool of predictive analytics. Will a loan default (yes or no)? Will it prepay? Will a web visitor click on a link? Will they purchase something? Is an insurance claim fraudulent? Often in classification problems, one class is of primary interest (e.g., the fraudulent insurance claim), and in binary classification, this class is designated as a 1, with the other, more prevalent class being a 0. Often, a key part of the process is estimating a *propensity score*, a probability of belonging to the class of interest. A common scenario is one in which the class of interest is relatively rare. When evaluating a classifier, there are a variety of model assessment metrics that go beyond simple accuracy; these are important in the rare-class situation, when classifying all records as 0s can yield high accuracy.