

Logistic Regression

Introduction

► Code

So far we have seen linear regression: a continuous valued observation is estimated as a linear (or affine) function of the independent variables.

Today the observation is a 0/1 variable and we want a *probability*:

- fit a linear model to the **log-odds** and invert it through the sigmoid,
- fit the coefficients by **maximum likelihood**,
- evaluate the result as a **classifier**, and choose its threshold.

Regularization – a penalty on the size of β , for linear and logistic regression alike – follows in [Lecture 11B](#).

Estimating a Probability

- Imagine that you are observing a binary variable – value 0 or 1.
- That is, these could be pass/fail, admit/reject, Democrat/Republican, etc.
- Assume there is some **probability** of observing a 1, and that probability is a function of certain independent variables.
- So the key properties of a problem that make it appropriate for logistic regression are:
 - You are trying to predict a **categorical** variable
 - You want to estimate a **probability** of seeing a particular value of the categorical variable.

Example: Grad School Admission

Let's consider this question:

What is the probability I will be admitted to Grad School?

Let's see how variables, such as,

- *GRE* (Graduate Record Exam scores),
- *GPA* (grade point average), and
- prestige of the undergraduate institution

affect admission into graduate school.

Example continued

The response variable, admit/don't admit, is a binary variable.

So there are three predictor variables: **gre**, **gpa** and **rank**.

- We will treat the variables *gre* and *gpa* as continuous.
- The variable *rank* takes on the values 1 through 4 with 1 being the highest prestige.

 Rank is ordinal, and we feed it in as a number

Treating *rank* as a number assumes that each step down in prestige has the *same* effect on the log-odds. The alternative is one indicator column per rank (`C(rank)` in formula syntax), which costs two more coefficients. We keep the numeric version here; the categorical version is shown in [Appendix: Regression Formulas with statsmodels](#).

Example continued

The first rows of the data (400 applicants), and some summary statistics:

► Code

	admit	gre	gpa	rank
0	0	380	3.61	3
1	1	660	3.67	3
2	1	800	4.00	1
3	1	640	3.19	4
4	0	520	2.93	4
5	1	760	3.00	2
6	1	560	2.98	1
7	0	400	3.08	2

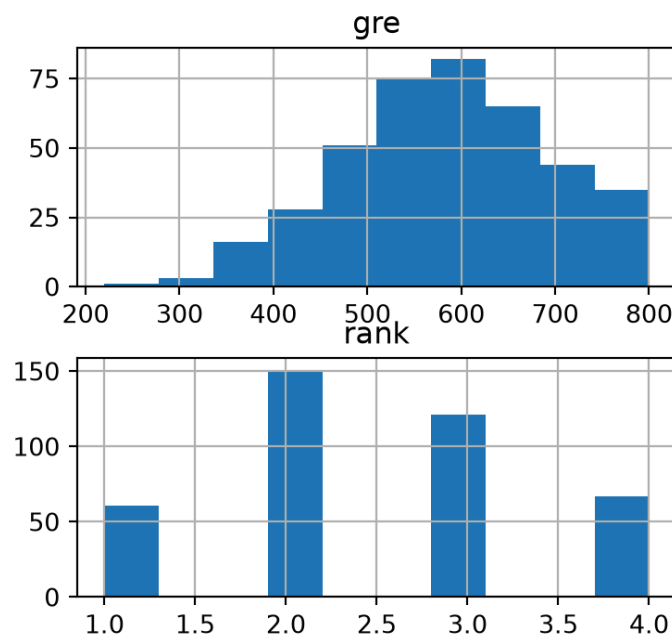
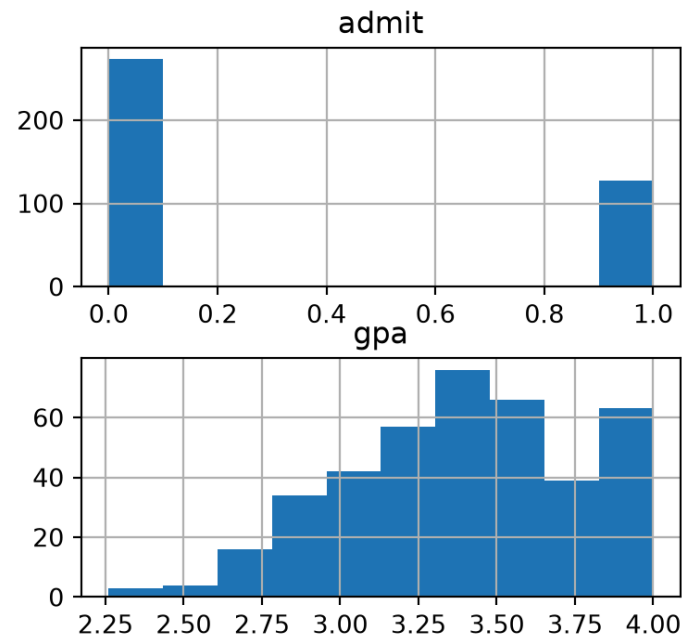
► Code

	admit	gre	gpa	rank
count	400.000	400.000	400.000	400.000
mean	0.318	587.700	3.390	2.485
std	0.466	115.517	0.381	0.944
min	0.000	220.000	2.260	1.000
25%	0.000	520.000	3.130	2.000
50%	0.000	580.000	3.395	2.000
75%	1.000	660.000	3.670	3.000
max	1.000	800.000	4.000	4.000

Example continued

We can also plot histograms of the variables:

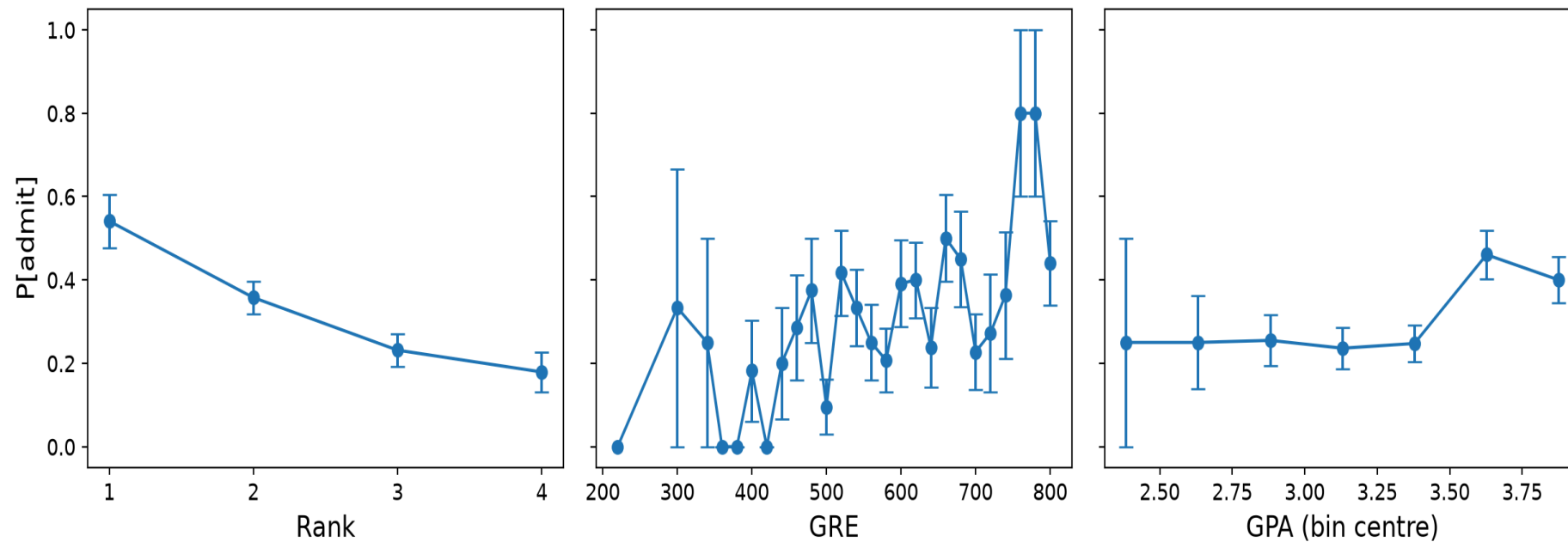
► Code



Example continued

Mean admission rate against each feature, with standard-error bars:

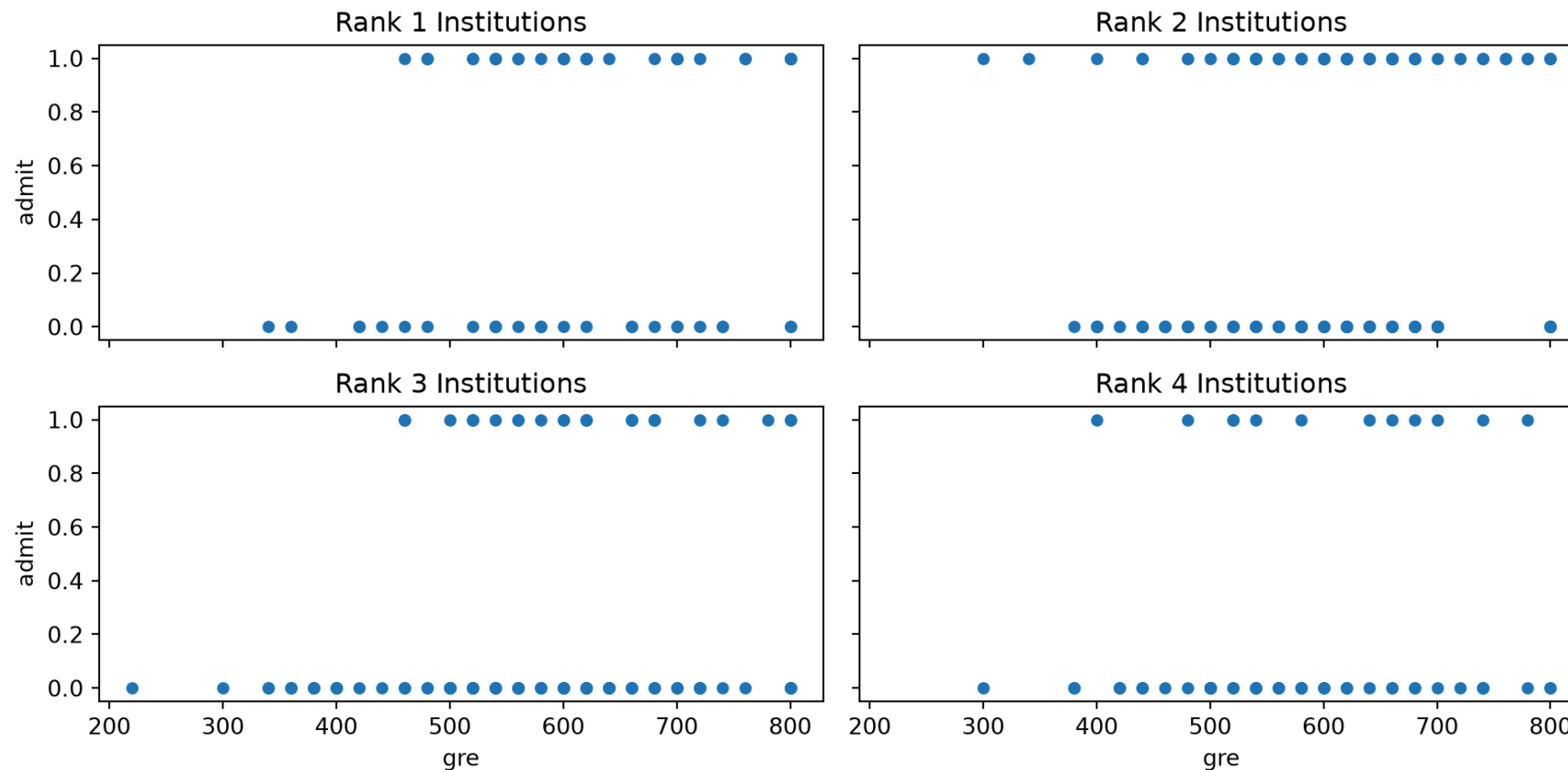
► Code



Example continued

Finally, we plot admission status versus GRE score for each data point for each of the four ranks:

► Code



Logistic Regression

- Logistic regression is concerned with estimating a **probability**.
- However, all that is available are categorical observations, which we will code as 0/1.
- That is, these could be pass/fail, admit/reject, Democrat/Republican, etc.
- Now, a linear function like $\beta_0 + \beta_1 x$ cannot be used to predict probability directly, because
 - the linear function takes on all values (from $-\infty$ to $+\infty$),
 - and probability only ranges over $[0, 1]$.

Odds and Log-Odds

However, there is a transformation of probability that works: it is called **log-odds**.

For any probability p , the **odds** is defined as $p/(1 - p)$, which is the ratio of the probability of an event to the probability of the non-event.

Notice that odds vary from 0 to ∞ , and odds < 1 indicates that $p < 1/2$.

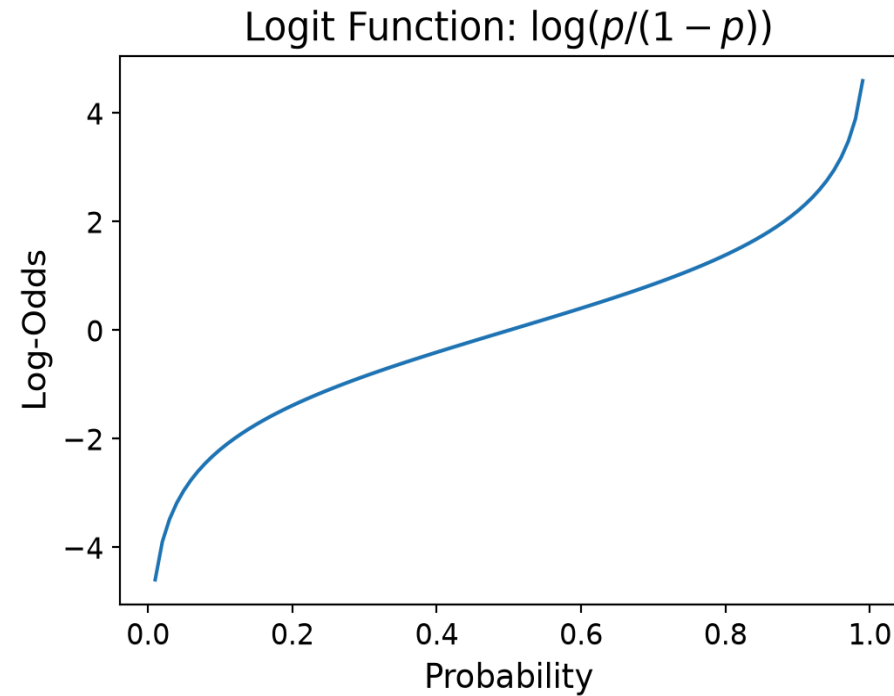
Odds are still not a good target for a *linear* function, for two reasons.

- **Range.** A line takes negative values; odds cannot. Taking the log maps $(0, \infty)$ onto the whole real line, so nothing the line predicts is impossible.
- **Symmetry.** Odds treat the two outcomes asymmetrically: $p = 0.9$ gives odds 9 while $p = 0.1$ gives odds $1/9$. Log-odds are symmetric: $+2.2$ and -2.2 . Swapping which outcome we call “1” just flips the sign of every coefficient.

Odds and Log-Odds

It stretches the interval $(0, 1)$ onto the whole real line, with $p = 1/2$ at 0: exactly the range a line can cover.

► Code



Odds and Log-Odds

So the logistic regression model says that the log-odds of $y = 1$ is a **linear** function of x , a straight line $\beta_0 + \beta_1 x$.

Solving for $p(y=1 \mid x)$:

$$\beta_0 + \beta_1 x = \log \frac{p(y=1 \mid x)}{1 - p(y=1 \mid x)}$$

$$e^{\beta_0 + \beta_1 x} = \frac{p(y=1 \mid x)}{1 - p(y=1 \mid x)} \quad (\text{exponentiate both sides})$$

$$e^{\beta_0 + \beta_1 x} (1 - p(y=1 \mid x)) = p(y=1 \mid x) \quad (\text{clear the denominator})$$

$$e^{\beta_0 + \beta_1 x} = p(y=1 \mid x) + p(y=1 \mid x) e^{\beta_0 + \beta_1 x} \quad (\text{expand})$$

$$\frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}} = p(y=1 \mid x) \quad (\text{collect terms})$$

Odds and Log-Odds: what β_1 means

The second line of the derivation says the odds are an exponential function of x :

$$\frac{p(y=1 \mid x)}{1 - p(y=1 \mid x)} = e^{\beta_0 + \beta_1 x} = e^{\beta_0} (e^{\beta_1})^x.$$

So each unit increase in x **multiplies the odds by the same factor** e^{β_1} , whatever the starting point:

$$\frac{\text{odds}(x + 1)}{\text{odds}(x)} = e^{\beta_1}.$$

Odds and Log-Odds

So, logistic regression fits a probability of the following form:

$$p(y=1 \mid x) = \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}}.$$

This is a **sigmoid function**; when $\beta_1 > 0$,

- as $x \rightarrow \infty$, then $p(y=1 \mid x) \rightarrow 1$ and
- as $x \rightarrow -\infty$, then $p(y=1 \mid x) \rightarrow 0$.

We plot this on the next slide.

Notation: $\sigma(z)$ and $\beta^T \mathbf{x}$

This type of sigmoid comes up so often that it gets its own symbol:

$$\sigma(z) = \frac{e^z}{1 + e^z} = \frac{1}{1 + e^{-z}}, \quad \text{so} \quad p(y=1 \mid x) = \sigma(\beta_0 + \beta_1 x).$$

and is sometimes called the **logistic function**.

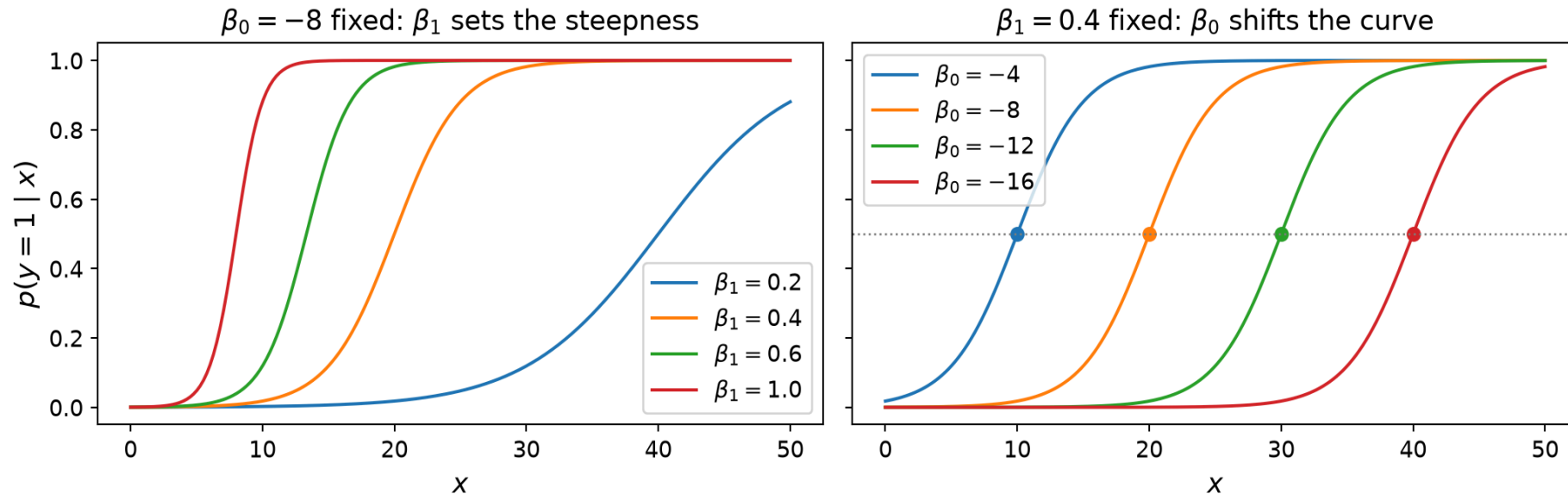
With d features we collect them, together with a leading 1 for the intercept, into $\mathbf{x} = (1, x_1, \dots, x_d)^T$ and the coefficients into $\beta = (\beta_0, \beta_1, \dots, \beta_d)^T$. The model is then

$$p(y=1 \mid \mathbf{x}) = \sigma(\beta^T \mathbf{x}), \quad \text{logit } p(y=1 \mid \mathbf{x}) = \beta^T \mathbf{x}.$$

Everything below is written for one feature; replace $\beta_0 + \beta_1 x$ by $\beta^T \mathbf{x}$ and it holds for any number of features.

What β_1 and β_0 do to the curve

► Code



- β_1 controls how fast $p(y=1 | x)$ rises from 0 to 1 (and its sign sets the direction).
- The curve crosses $p = 1/2$ where the log-odds are zero, at $x = -\beta_0/\beta_1$;
 - increasing β_0 shifts the curve to the left (the dots on the right panel).

Odds and Log-Odds

So we had the **logistic** sigmoid function:

$$\sigma(z) = \frac{e^z}{1 + e^z} = \frac{1}{1 + e^{-z}}, \quad \text{where} \quad p(y=1 \mid x) = \sigma(\beta_0 + \beta_1 x).$$

which maps $(-\infty, \infty)$ to $(0, 1)$.

The inverse of that is confusingly called the **logit** function:

$$\text{logit}(p) = \log \left(\frac{p}{1 - p} \right)$$

that maps $p \in (0, 1)$ to the real line $(-\infty, \infty)$.

So a compact way to describe logistic regression is that it finds regression coefficients β_0, β_1 to fit:

$$\text{logit}(p(y=1 \mid x)) = \log \left(\frac{p(y=1 \mid x)}{1 - p(y=1 \mid x)} \right) = \beta_0 + \beta_1 x.$$

Odds and Log-Odds

So, the best way to think of logistic regression is that we compute a linear function:

$$\beta_0 + \beta_1 x,$$

and then *map* that to a probability using the logistic function:

$$\sigma(\beta_0 + \beta_1 x) = \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}}.$$

Note

And again, in machine learning we just usually call this the **sigmoid** function, and denote it $\sigma(z)$.

Logistic vs Linear Regression

Let's take a moment to compare linear and logistic regression.

In **Linear regression** we fit

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i.$$

We do the fitting by minimizing the sum of squared errors $\|\epsilon\|^2$. This can be done in closed form using either geometric arguments or by calculus.

Now, if ϵ_i comes from a normal distribution with mean zero and some fixed variance, then minimizing the sum of squared errors is exactly the same as finding the maximum likelihood of the data with respect to the probability of the errors.

So, in the case of linear regression, it is a lucky fact that the **MLE** of β_0 and β_1 can be found by a **closed-form** calculation.

Logistic vs Linear Regression

In **Logistic regression** we fit

$$\text{logit}(p(y_i=1 \mid x_i)) = \beta_0 + \beta_1 x_i.$$

How should we choose parameters?

Here too, we use Maximum Likelihood Estimation of the parameters.

That is, we choose the parameter values that maximize the likelihood of the data given the model.

$$P(y_i \mid x_i) = \begin{cases} \sigma(\beta_0 + \beta_1 x_i) & \text{if } y_i = 1 \\ 1 - \sigma(\beta_0 + \beta_1 x_i) & \text{if } y_i = 0 \end{cases}$$

Logistic vs Linear Regression

We can write this as a single expression. Writing $p_i = p(y_i=1 \mid x_i) = \sigma(\beta_0 + \beta_1 x_i)$,

$$P(y_i \mid x_i) = p_i^{y_i} (1 - p_i)^{1-y_i},$$

where we assume the parameters are fixed.

We can reinterpret this to express the **likelihood** of parameters β_0, β_1 given that we have observed the data (x_i, y_i) :

$$L(\beta_0, \beta_1 \mid x_i, y_i) = p_i^{y_i} (1 - p_i)^{1-y_i}.$$

That is one observation. We have n of them.

From one observation to the whole data set

The observations are independent, so the likelihood of the data set is the **product** and its logarithm – the **log-likelihood** – is the sum:

$$L(\beta) = \prod_{i=1}^n p_i^{y_i} (1 - p_i)^{1-y_i} \quad \Rightarrow \quad \ell(\beta) = \sum_{i=1}^n [y_i \log p_i + (1 - y_i) \log(1 - p_i)].$$

- $-\ell(\beta)/n$ is the **log-loss** (also called *cross-entropy*): the quantity [sklearn](#) and every neural-network library report. Maximizing ℓ and minimizing log-loss are the same thing.
- **This is our objective function to maximize.** Unlike least squares it is not quadratic in β , so setting the gradient to zero gives no closed form – we climb numerically.
- **There is a unique maximum** because the log-likelihood is concave in β .

From Scalar to Vector Notation

Just like before, we can create our design matrix of input observations with a leading column of 1s for the intercept, and the vector of output observations¹:

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \\ 1 & x_n \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix},$$

And our initialized $\beta^{(0)} = [0, 0]^T$.

1. Shown here for 1 feature for simplicity.

From Scalar to Vector Notation

When computing the full vector of predicted probabilities $\mathbf{p}$:

$$\mathbf{p} = \sigma(X\beta)$$

1. $X\beta$ yields an $n \times 1$ vector of scalar log-odds (one $z_i = \beta^T \mathbf{x}_i$ per observation).
2. $\sigma(\cdot)$ is then applied element-by-element to that vector:

$$\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{bmatrix} = \begin{bmatrix} \sigma(\beta^T \mathbf{x}_1) \\ \sigma(\beta^T \mathbf{x}_2) \\ \vdots \\ \sigma(\beta^T \mathbf{x}_n) \end{bmatrix} = \begin{bmatrix} \frac{1}{1+e^{-\beta^T \mathbf{x}_1}} \\ \frac{1}{1+e^{-\beta^T \mathbf{x}_2}} \\ \vdots \\ \frac{1}{1+e^{-\beta^T \mathbf{x}_n}} \end{bmatrix}$$

This maps every linear prediction z_i into a valid probability $p_i \in (0, 1)$.

The gradient of the log-likelihood

To calculate the gradient of the log-likelihood, we need the derivative of the sigmoid function.

Fortunately, the sigmoid has a derivative that is easy to remember:

$$\sigma'(z) = \sigma(z) (1 - \sigma(z)).$$

Using it, with $p_i = \sigma(\beta^T \mathbf{x}_i)$, where $\mathbf{x}_i$ is the i^{th} row of the design matrix, and using the chain rule to break apart the derivative:

$$\frac{\partial \ell_i}{\partial \beta} = \frac{\partial \ell_i}{\partial p_i} \cdot \frac{\partial p_i}{\partial z_i} \cdot \frac{\partial z_i}{\partial \beta}$$

the partial derivative of the log-likelihood for a single observation is:

$$\nabla \ell(\beta) = \sum_{i=1}^n (y_i - p_i) \mathbf{x}_i = X^T(\mathbf{y} - \mathbf{p}),$$

where X is the $n \times (p + 1)$ matrix with rows $\mathbf{x}_i^T$ and $\mathbf{p} = \sigma(X\beta)$ applied entrywise.

Residual times feature – the same shape as the normal equations $X^T(\mathbf{y} - X\beta) = 0$ from linear regression, with a sigmoid inside. The difference is that $\mathbf{p}$ depends on β nonlinearly, so we cannot solve $\nabla \ell = 0$ directly.

The notes has the full derivation of the gradient.

Gradient ascent on the log-likelihood

Algorithm:

1. **Initialize** parameters: Start with random values $\beta^{(0)}$
2. **Compute gradient**: Calculate $\nabla \ell(\beta^{(t)}) = X^T(\mathbf{y} - \mathbf{p}^{(t)})$ – the direction of steepest increase
3. **Update parameters**: Take a step in that direction:

$$\beta^{(t+1)} = \beta^{(t)} + \eta \nabla \ell(\beta^{(t)})$$

where η is the **learning rate** (step size; the same η as in Lecture 16)

4. **Repeat** steps 2-3 until convergence (gradient ≈ 0 or max iterations reached)

Result: Parameters that maximize the likelihood of the observed data. The log-likelihood of logistic regression is concave, so this maximum is global.

We are climbing ℓ , so this is gradient *ascent*; the libraries minimize the log-loss instead, which is gradient *descent* on $-\ell$ – the same steps. Gradient descent gets a full treatment in [Neural Networks I](#).

One gradient step by hand

Three observations, one feature: $\mathbf{x} = [-1, 0, 2]$, $\mathbf{y} = [0, 0, 1]$, so adding a column of 1s for the intercept gives

$$\mathbf{X} = \begin{bmatrix} 1 & -1 \\ 1 & 0 \\ 1 & 2 \end{bmatrix}$$

Start at $\boldsymbol{\beta}^{(0)} = [0, 0]^T$ with $\eta = 0.1$.

We can evaluate $\mathbf{p} = \sigma(\mathbf{X}\boldsymbol{\beta}^{(0)})$ and the gradient $\nabla \ell(\boldsymbol{\beta}^{(0)})$ by hand.

- $\boldsymbol{\beta}^T \mathbf{x}_i = 0$ for every i , so $\mathbf{p} = (0.5, 0.5, 0.5)^T$ and $\mathbf{y} - \mathbf{p} = (-0.5, -0.5, 0.5)^T$.
- $\partial \ell / \partial \beta_0 = \sum_i (y_i - p_i) = -0.5$; $\partial \ell / \partial \beta_1 = \sum_i (y_i - p_i) x_i = 0.5 + 0 + 1.0 = 1.5$.
- $\boldsymbol{\beta}^{(1)} = (0, 0) + 0.1 (-0.5, 1.5) = (-0.05, 0.15)$.

The intercept moves down (two of three labels are 0) and the slope moves up (the 1 sits at the largest x). In today's activity you will code exactly this loop and check that it converges to what [sklearn](#) reports.

Logistic Regression In Practice

So, in summary, we have:

Input pairs $(\mathbf{x}_i, y_i)$

Output parameters $\hat{\beta}$ that maximize the likelihood of the data given these parameters for the logistic regression model.

Method Maximum likelihood estimation, obtained by gradient ascent on ℓ (the packages use a faster second-order variant of the same idea).

The standard package will give us a coefficient β_j for each independent variable (feature).

Logistic Regression in Practice

If we want to include a constant (i.e., β_0) we need to add a column of 1s (just like in linear regression).

► Code

```
Index(['gre', 'gpa', 'rank', 'intercept'], dtype='str')
```

► Code

```
Optimization terminated successfully.  
Current function value: 0.574302  
Iterations 6
```

Logistic Regression in Practice

Statsmodels gives us a summary of the model fit.

► Code

Logit Regression Results

Dep. Variable:	admit	No. Observations:	400
Model:	Logit	Df Residuals:	396
Method:	MLE	Df Model:	3
Date:	Sat, 17 Oct 2026	Pseudo R-squ.:	0.08107
Time:	12:44:29	Log-Likelihood:	-229.72
converged:	True	LL-Null:	-249.99
Covariance Type:	nonrobust	LLR p-value:	8.207e-09

	coef	std err	z	P> z	[0.025	0.975]
gre	0.0023	0.001	2.101	0.036	0.000	0.004
gpa	0.7770	0.327	2.373	0.018	0.135	1.419
rank	-0.5600	0.127	-4.405	0.000	-0.809	-0.311
intercept	-3.4495	1.133	-3.045	0.002	-5.670	-1.229

All three predictors are significant at the 5% level: no 95% confidence interval contains zero. The GRE interval is displayed as `[0.000, 0.004]` only because of rounding; unrounded it is

Reading the coefficients

The coefficients live on the **log-odds** scale, and e^{β_j} is the factor by which the odds are multiplied when x_j increases by one unit, holding the other features fixed.

► Code

	coefficient	odds ratio e^{β}
gre	0.002	1.002
gpa	0.777	2.175
rank	-0.560	0.571
intercept	-3.450	0.032
gre (per 100 points)	0.229	1.258

- **GPA:** $\hat{\beta} = 0.777$, so one more GPA point multiplies the odds of admission by $e^{0.777} \approx 2.17$ – it more than doubles them.
- **GRE:** $\hat{\beta} = 0.0023$ per point, which is 0.23 per 100 points, so 100 more GRE points multiply the odds by $e^{0.23} \approx 1.26$.
- **Rank:** $\hat{\beta} = -0.56$, so each step *down* in prestige (rank 1 to 2, 2 to 3, ...) multiplies the odds by $e^{-0.56} \approx 0.57$ – nearly halves them.

Reading the intercept

► Code

```
intercept beta_0 = -3.450;    sigma(beta_0) = 0.031  
fraction admitted in the data = 0.3175  
p(y=1 | x) at the mean GRE, GPA, rank = 0.298
```

- $\sigma(\hat{\beta}_0) = 3.1\%$ is the model's admission probability for a student with GRE 0, GPA 0.0 and rank 0. It is **not** the base rate: 31.75% of the applicants were admitted.
- Had we centered the features first, $x = 0$ would be the *average* applicant, and $\sigma(\hat{\beta}_0)$ would be close to the admit rate (0.30 here).

Using the Model

Note that by fitting a model to the data, we can make predictions for inputs that were not in the training data.

Furthermore, we can make a prediction of a probability for cases where we don't have enough data to estimate the probability directly – e.g., for specific parameter values.

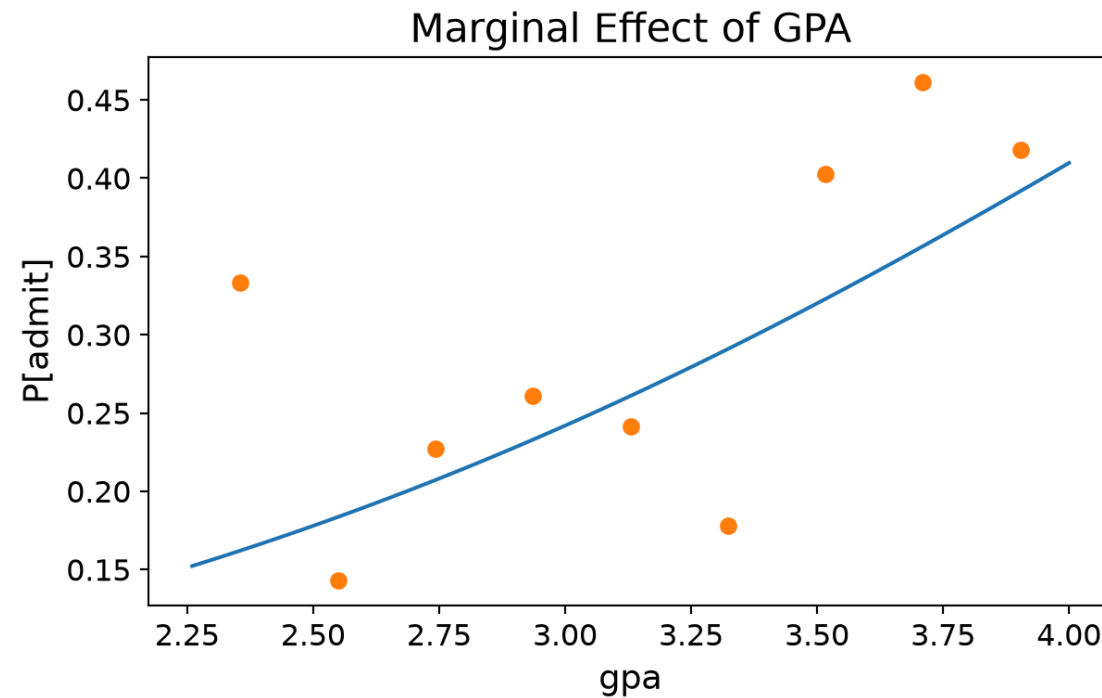
Let's see how well the model fits the data.

Using the Model

We have three independent variables, so in each case we'll use average values for the two that we aren't evaluating.

GPA (GRE = 600, Rank = 2.5):

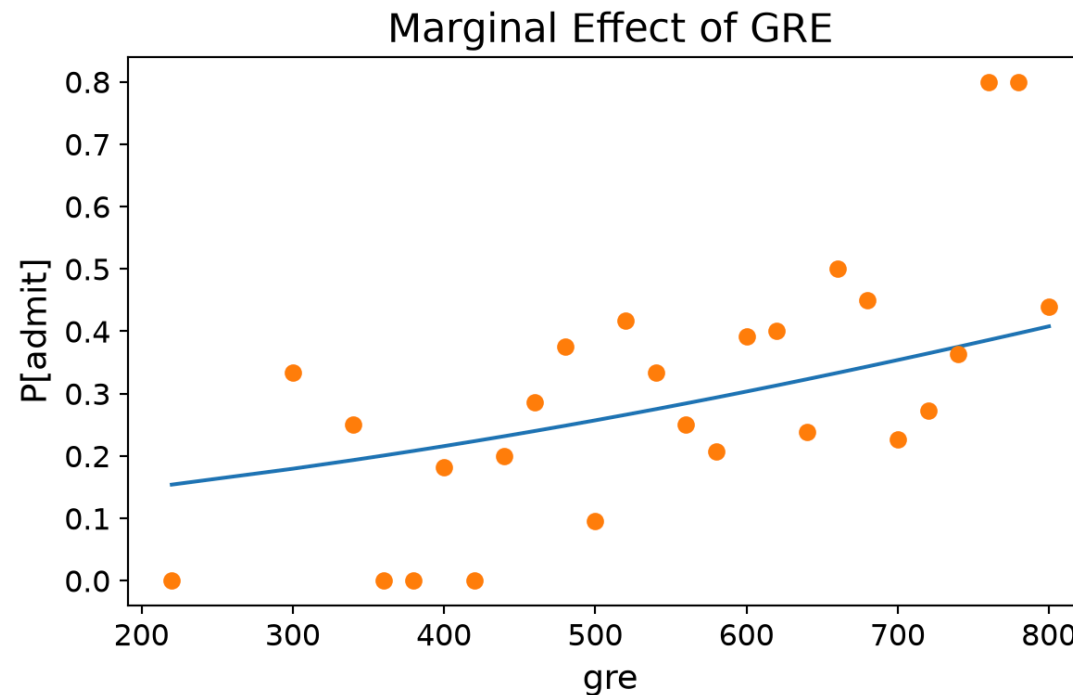
► Code



Logistic Regression in Practice

GRE Score (GPA = 3.4, Rank = 2.5):

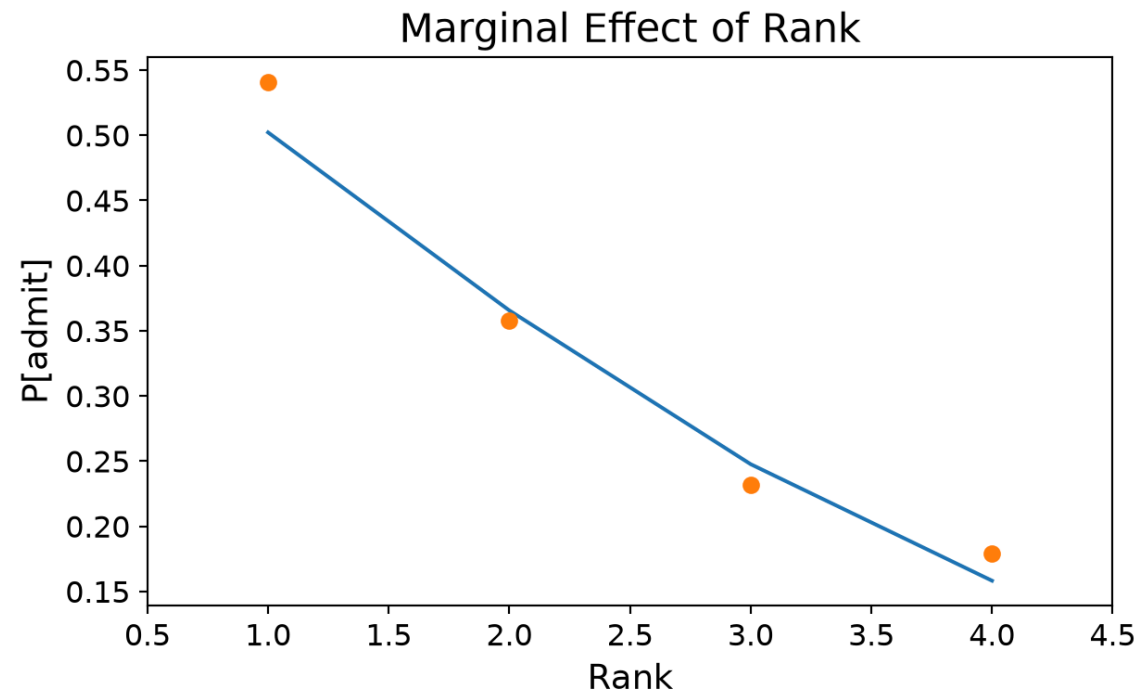
► Code



Logistic Regression in Practice

Institution Rank (GRE = 600, GPA = 3.4):

► Code



Logistic Regression in Perspective

At the start of lecture we emphasized that logistic regression is concerned with estimating a **probability** model for **discrete** (0/1) data.

However, it may well be the case that we want to do something with the probability that amounts to **classification**.

For example, we may classify data items using a rule such as “Assign item $\mathbf{x}_i$ to Class 1 if $p(y_i=1 \mid \mathbf{x}_i) > 0.5$ ”.

For this reason, logistic regression could be considered a classification method.

The decision boundary is still a hyperplane

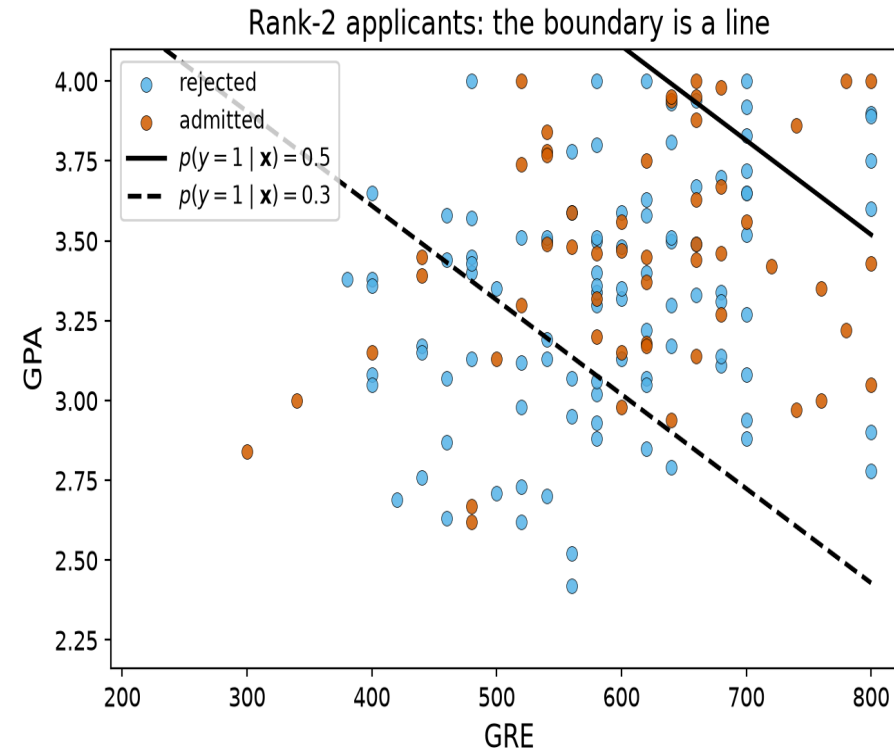
- The rule “ $p(y=1 \mid \mathbf{x}) > 0.5$ ” is the rule “ $\beta^T \mathbf{x} > 0$ ”
- the sigmoid is monotonic, so thresholding the probability is the same as thresholding the *linear* score.
- The boundary $\beta^T \mathbf{x} = 0$ is a **hyperplane** – a line in two features – exactly as in the linear classifiers you have seen.
- The nonlinearity lives in the *confidence*, not in the boundary.

Any other threshold t slides the hyperplane parallel to itself, to $\beta^T \mathbf{x} = \text{logit}(t)$.

The decision boundary on the admissions data

Rank-2 applicants, with the boundary $\hat{\beta}^T \mathbf{x} = \text{logit}(t)$ drawn for $t = 0.5$ and $t = 0.3$ (solved for GPA as a function of GRE):

► Code



Above the line the model says “admit”. At $t = 0.5$ almost no rank-2 applicant qualifies; lowering the threshold to 0.3 slides the same line down into the cloud. Changing the threshold never bends the boundary.

Logistic Regression as a classifier

Let's use our logistic regression as a classifier.

We want to ask whether we can correctly predict whether a student gets admitted to graduate school.

Let's separate our training and test data:

```
1 X_train, X_test, y_train, y_test = model_selection.train_test_split(  
2     df[train_cols], df['admit'],  
3     test_size=0.4, random_state=1)
```

Logistic Regression in Perspective

Now, remember from precision and recall, we can consider the trade-offs.

Let's say our classifier is outputting “yes” when it thinks the student will be admitted.

There are four cases:

- Classifier says “yes”, and student **is** admitted: **True Positive**.
- Classifier says “yes”, and student **is not** admitted: **False Positive**.
- Classifier says “no”, and student **is** admitted: **False Negative**.
- Classifier says “no”, and student **is not** admitted: **True Negative**.

Accuracy on imbalanced data

Accuracy is the fraction of all classifications that are correct: $(TP + TN)/n$.

Accuracy has a floor set by the **prevalence** (the base rate).

Only 31.75% of these applicants were admitted, so a “classifier” that says *no* to everyone is **68% accurate** without looking at a single feature.

Under class imbalance, accuracy mostly measures the imbalance – which is why we need the next two metrics.

Precision and Recall Refresher

Precision is the fraction of “yes” classifications that are correct:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}.$$

Recall is the fraction of admits that we say “yes” to:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}.$$

► Code

```
Precision: 0.586, Recall: 0.340, Accuracy: 0.719  
Accuracy of "always reject" on this test set: 0.688
```

The accuracy barely beats “always reject”, and the recall says why: at a 0.5 threshold the model says yes to only a small fraction of the real admits.

Logistic Regression in Perspective

A single random split is noisy. Let's average precision and recall over 20 different train/test splits:

► Code

► Code

Average Precision: 0.612, Average Recall: 0.258, Average Accuracy: 0.712

Logistic Regression in Perspective

Sometimes we would like a single value that describes the overall performance of the classifier.

For this, we take the *harmonic mean* of precision and recall, called **F1 Score**:

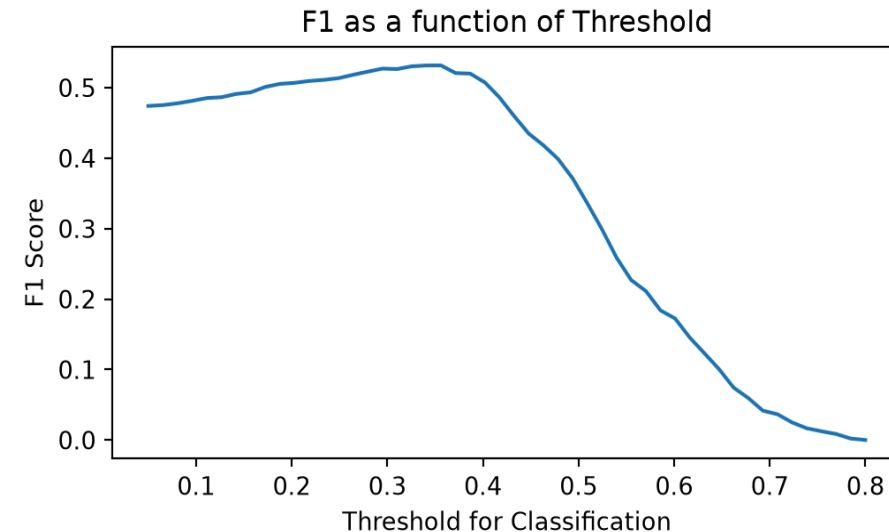
$$\text{F1 Score} = \frac{2}{\text{Recall}^{-1} + \text{Precision}^{-1}} = 2 \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

F1 Score as a function of threshold

Using this, we can evaluate other settings for the threshold.

► Code

► Code



Based on this plot, we can say that the best classification threshold appears to be around 0.35 – well below 0.5 – where precision and recall are:

► Code

Precision: 0.464, Recall: 0.618, F1: 0.530

Compared with the 0.5 threshold we traded precision for recall: the model now says yes to most of the real admits, at the cost of more false alarms.

The threshold is a decision

0.5 is not a law of nature: it is where the model thinks the two classes are equally likely, which says nothing about what the two kinds of mistake cost.

If a false positive costs C_{FP} and a false negative costs C_{FN} , saying “yes” is worth it exactly when $p C_{FN} > (1 - p) C_{FP}$, i.e. when

$$p > t^* = \frac{C_{FP}}{C_{FP} + C_{FN}}.$$

A miss that is four times as costly as a false alarm puts the threshold at $t^* = 0.2$.

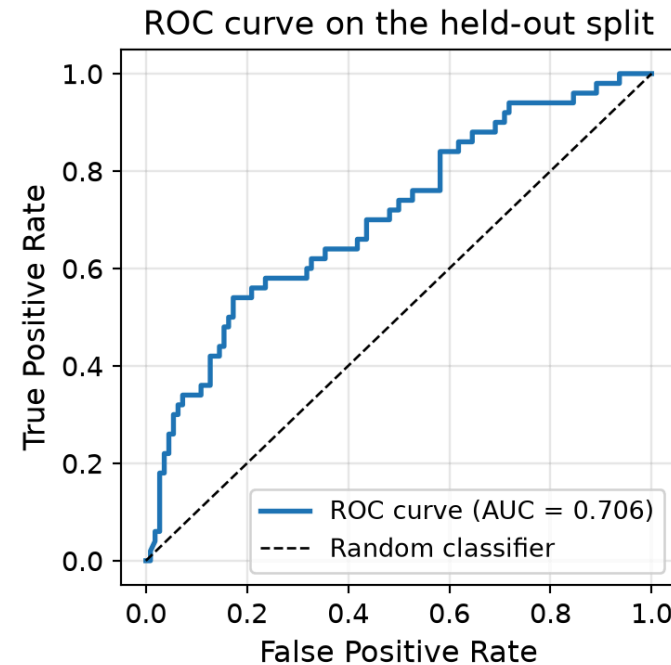
Choosing t is a decision about costs, and F1 is just one cost-free way to summarize the trade-off.

ROC-AUC Curve

Two rates, both out of the *true* classes:

- **True positive rate** (TPR, = recall): $TP / (TP + FN)$, the fraction of real admits we say yes to.
- **False positive rate** (FPR): $FP / (FP + TN)$, the fraction of real rejects we wrongly say yes to.
- The **ROC curve** plots TPR against FPR, one point per threshold. The diagonal is a random classifier.
- The **AUC** is the area under the ROC curve: a threshold-free summary of how well the model *ranks* admits above rejects.

► Code



AUC Score: 0.7061

Recap: what you can now do

1. **Write the logistic model** – a linear function $\beta^T \mathbf{x}$ equal to the log-odds, inverted through the sigmoid σ – read β_j as a log-odds slope and e^{β_j} as an odds ratio, and explain why the decision boundary is still a hyperplane (the nonlinearity lives in the confidence).
2. **Fit it by maximum likelihood**: write the log-likelihood (minus the log-loss), use $\sigma' = \sigma(1 - \sigma)$ to get the gradient $X^T(\mathbf{y} - \mathbf{p})$, take one gradient step by hand, and say why there is no closed form.
3. **Evaluate a classifier beyond accuracy**: accuracy has a floor set by prevalence; sweep the threshold, trade precision against recall and F1, and read ROC/AUC for what it does (ranking) and does not (choose a threshold) tell you.
4. **Choose a threshold to ship**: the cost-optimal threshold is $C_{FP}/(C_{FP} + C_{FN})$ – in today's activity you will sweep precision, recall and F1 on a held-out split of an imbalanced loan-default problem, read its ROC-AUC, and compare the cost-optimal threshold with 0.5 and with `class_weight="balanced"`.

Next: Regularization (Lecture 11B)

- Linear and logistic models can both **overfit**: with many or correlated features, $\hat{\beta}$ chases noise in the training set.
- The fix is a **penalty** on the size of β : ridge and LASSO.
- [Lecture 11B](#) develops it for linear regression and applies it to today's logistic model.