BUSN 20800: Big Data

Lecture 2: Regression

Dacheng Xiu

University of Chicago Booth School of Business

Regression Revisited

Regression through linear models, and how to do it in Python.

Interaction, factor effects, design (`model`) matrices.

Logistic Regression: an essential BD tool.

Estimation: Maximum Likelihood and Minimum Deviance

Implementation: Gradient Descent

Much of this should be review, but emphasis will be different.

Linear Models

Many problems in BD involve a response of interest ('y') and a set of covariates ('**x**') to be used for prediction.

We'll model the conditional mean for y given **x**,

$$\mathrm{E}[\ y \mid \mathbf{x}\ ] = f(\mathbf{x}'\boldsymbol{\beta})$$

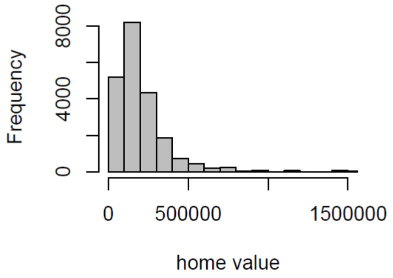
$\mathbf{x} = [1, x_1, x_2, \ldots x_p]$ is your vector of covariates.

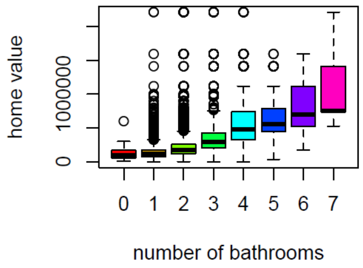
$\boldsymbol{\beta} = [\beta_0, \beta_1, \beta_2, \ldots \beta_p]$ are the corresponding coefficients.

The product is $\mathbf{x}'\boldsymbol{\beta} = \beta_0 + x_1\beta_1 + x_2\beta_2 + \ldots + x_p\beta_p$.

For notational convenience we use $x_0 = 1$ for the intercept.

Marginal and conditional distributions





On the left, all of the homes are grouped together.
On the right, home prices are grouped by # baths.

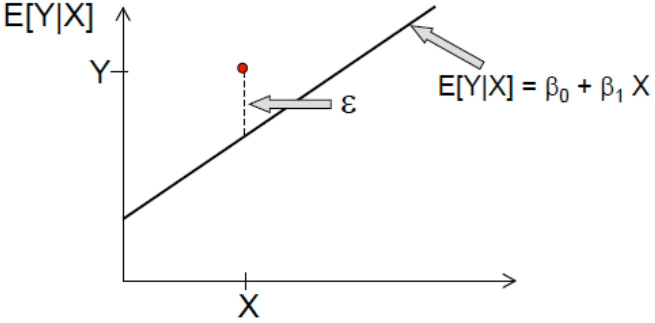
The marginal mean is a simple number.
The conditional mean is a function that depends on covariates.
The data is distributed randomly around these means.

In a Gaussian linear regression,

$$y \mid \mathbf{x} \sim \mathrm{N}(\mathbf{x}'\boldsymbol{\beta}, \sigma^2)$$

Conditional mean is $\mathrm{E}[y|\mathbf{x}] = \mathbf{x}'\boldsymbol{\beta}$. Conditional variance is $\mathrm{Var}[y|\mathbf{x}] = \sigma^2$.

A common way of writing the regression model is $y = \mathbf{x}'\boldsymbol{\beta} + \varepsilon$. With just a single variable x, we have a simple geometric representation of data.



Orange Juice

Three brands (b) Tropicana, Minute Maid, Dominicks

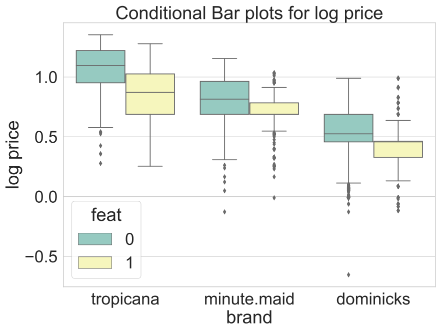
83 Chicagoland Stores
Demographic info for each

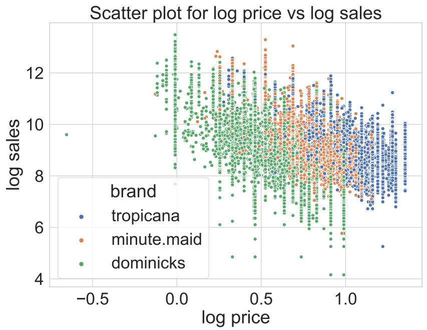
Price, sales (log of # units sold), and whether advertised (`feat`)

data in `oj.csv`

bayesm & Montgomery, 1987

The Juice: price, brand, and sales

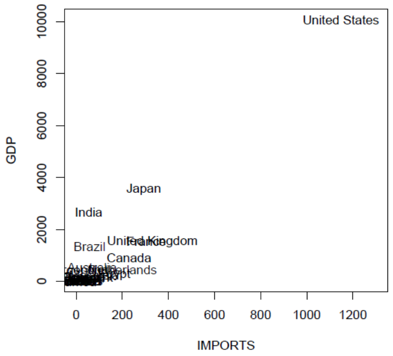


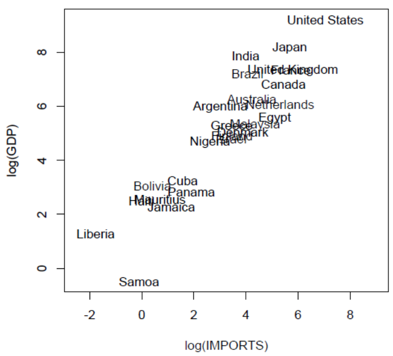


Each brand occupies a well defined price range. Sales decrease with price.

Thinking About Scale

When making a linear point (this goes up, that goes down) think about the scale on which you expect to find linearity.





If your scatterplots look like the left panel, consider using log. Outliers?

log linear

We often model the mean for log(y) instead of y.

Why? Multiplicative (rather than additive) change.

$\log(y) = \log(a) + x\beta \Leftrightarrow y = ae^{x\beta}$.

Predicted y is multiplied by e^{β} after a unit increase in x.

Recall that $\log(y) = z \Leftrightarrow e^{z} = y$ where $e \approx 2.7$

$\log(ab) = \log(a) + \log(b)$ and $\log(a^{b}) = b\log(a)$.

I use log=ln, natural log. Anything else will be noted, e.g., $\log_2$.

Whenever y changes on a percentage scale, use log(y).

prices: "... Foreclosed homes sell at a 20% to 30% discount"

sales: "... our y.o.y. sales are up 20% across models"

Price Elasticity

A simple orange juice 'elasticity model' for sales y has

$$\mathbb{E}[\log y] = \gamma \log(\texttt{price}) + \mathbf{x}'\boldsymbol{\beta}$$

Elasticities and log-log regression: for small values we can interpret γ as % change in y per 1% increase in `price`.

We run this in Python:

```
smf.glm(formula='log_sales ~  log_price+brand',data=oj).fit()
  (Intercept) log(price) branBDinute.maid brandtropicana
   10.8288    -3.1387           0.8702         1.5299
```

and see sales drop by about 3.1% for every 1% price hike.

Regression in Python

You need only one command

```
reg = smf.glm(y ~  var1 + ...  + varP, data=mydata).fit()
```

glm stands for Generalized Linear Model.

`y ~ a + b` is the 'formula' that defines your regression.

The object `reg` is a list of useful things.

`reg.summary()` prints a bunch of information.

`reg.params` gives coefficients.

`reg.predict(mynewdata)` predicts.

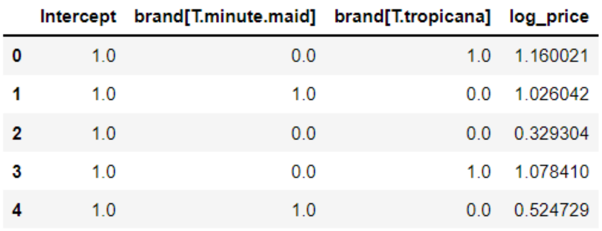
`mynewdata` must be a data frame with exactly the same format as `mydata` (same variable names, same factor levels).

The Design Matrix

What happened to `branddominicks`?

Our regression formulas look like $\beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots$
But `brand` is not a number, so you can't do `brand`$\times\beta$.

The first step of `glm` is to create a numeric *design matrix*.
It does this with a call to the `dmatrix` function:



	Intercept	brand[T.minute.maid]	brand[T.tropicana]	log_price
0	1.0	0.0	1.0	1.160021
1	1.0	1.0	0.0	1.026042
2	1.0	0.0	0.0	0.329304
3	1.0	0.0	1.0	1.078410
4	1.0	1.0	0.0	0.524729

The dummy variable is on the left, and on the right we have numeric x that we can multiply against β coefficients.

Intercepts

The OJ `glm` used `dmatrix` to build this 4 column design:

```
dmatrix(' ~ log_price + brand', data = oj)
Intercept  branBDinute.maid    brandtropicana   log(price)
1.00000      0.000000             1.000000      1.160021
1.00000      1.000000             0.000000      1.026042
1.00000      0.000000             0.000000      0.329304
1.00000      0.000000             1.000000      1.078410
1.00000      1.000000             0.000000      0.524729
```

Each factor's reference level is absorbed by the intercept. Coefficients are 'change relative to reference' (dominicks here).

Why not include all three dummies?

Interaction

Beyond additive effects: variables change how others act on y.

An interaction term is the product of two covariates,

$$\mathbb{E}[\, y \mid \mathbf{x} \,] = \ldots + \beta_j x_j + x_j x_k \beta_{jk}$$

so that the effect on $\mathbb{E}[y]$ of a unit increase in x_j is $\beta_j + x_k \beta_{jk}$. It depends upon x_k!

Interactions play a massive role in statistical learning, and they are often central to social science and business questions.

- Does gender change the effect of education on wages?
- Do patients recover faster when taking drug A?
- How does advertisement affect price sensitivity?

Fitting interactions in Python: **use * in your formula.**

```
reg_interact = smf.glm(formula='log_sales ~
        log_price * brand', data=oj).fit()
```

```
Coefficients:
                (Intercept)                    log(price)
                   10.95468                      -3.37753
           branBDinute.maid               brandtropicana
                    0.88825                       0.96239
log(price):branBDinute.maid log(price):brandtropicana
                    0.05679                       0.66576
```

This is the model $\mathbb{E}[\log(v)] = \alpha_b + \beta_b \log(\texttt{price})$:
a separate intercept and slope for each brand 'b'.

Why not separately estimate three models?

Elasticities are dominicks: -3.4, minute maid: -3.3, tropicana: -2.7.

Where do these numbers come from? Do they make sense?

Advertisements

A key question: what changes when we feature a brand?
Here, this means in-store display promo or flier ad.

You could model the additive effect on log sales volume

$$\mathbb{E}[\log(\mathrm{v})] = \alpha_b + \mathbb{1}_{[\mathtt{feat}]}\alpha_{\mathtt{feat}} + \beta_b \log(\mathrm{p})$$

Or this *and* its effect on elasticity

$$\mathbb{E}[\log(v)] = \alpha_b + \beta_b \log(\mathrm{p}) + \mathbb{1}_{[\mathtt{feat}]}\left(\alpha_{\mathtt{feat}} + \beta_{\mathtt{feat}} \log(\mathrm{p})\right)$$

Or its *brand-specific* effect on elasticity

$$\mathbb{E}[\log(v)] = \alpha_b + \beta_b \log(\mathrm{p}) + \mathbb{1}_{[\mathtt{feat}]}\left(\alpha_{b,\mathtt{feat}} + \beta_{b,\mathtt{feat}} \log(\mathrm{p})\right)$$

See the Python code for runs of all three models.
Connect the regression formula and output to these equations.

Brand-specific elasticities

	Dominicks	*Minute Maid*	*Tropicana*
Not Featured	-2.8	-2.0	-2.0
Featured	-3.2	-3.6	-3.5

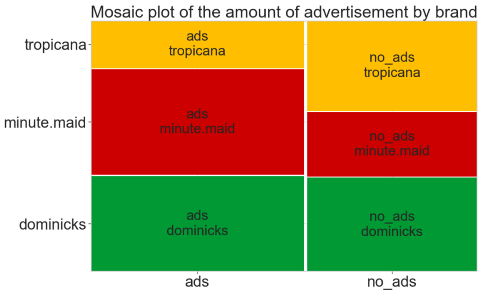
Ads always decrease elasticity.

Minute Maid and Tropicana elasticities drop 1.5% with ads, moving them from less to more price sensitive than Dominicks.

Why does marketing increase price sensitivity?
And how does this influence pricing/marketing strategy?

Confounding

Before including `feat`, Minute Maid behaved like Dominicks. With `feat`, Minute Maid looks more like Tropicana. Why?



Because Minute Maid was more heavily promoted, and promotions have a negative effect on elasticity, we were *confounding* the two effects in the brand average elasticity.

Logistic Regression

Linear regression is just one type of linear model.

Logistic regression: when y is true or false $(1/0)$.

Binary response as a prediction target:

- Profit or Loss, greater or less than, Pay or Default.
- Thumbs up or down, buy or not buy, potential customer?
- Win or Lose, Sick or Healthy, Republican or Democrat.

Building a linear model for binary response data

Recall our original model specification: $\mathbb{E}[\, y \mid \mathbf{x}\,] = f(\mathbf{x}'\boldsymbol{\beta})$.

The response 'y' is 0 or 1, leading to conditional mean:

$$\mathbb{E}[y|\mathbf{x}] = p(y=1|\mathbf{x}) \times 1 + p(y=0|\mathbf{x}) \times 0 \; = p(y=1|\mathbf{x}).$$

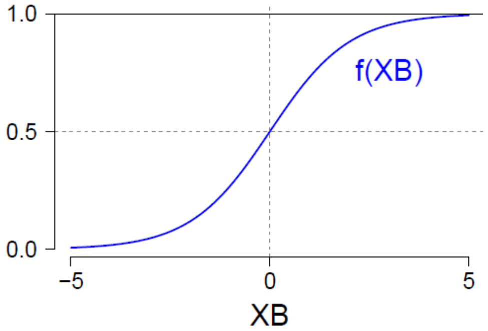
$\Rightarrow$ The expectation is a probability.

We'll choose $f(\mathbf{x}'\boldsymbol{\beta})$ to give values between zero and one.

We want a binary choice model

$$p = P(\ y = 1 \mid \mathbf{x}\ ) = f(\ \beta_0 + \beta_1 x_1 \ldots + \beta_p x_p\ )$$

where f is a function that increases in value from zero to one.



We'll use the logit link and do '**logistic regression**'.

$$P(y=1|\mathbf{x}) = \frac{e^{\mathbf{x}'\boldsymbol{\beta}}}{1+e^{\mathbf{x}'\boldsymbol{\beta}}} = \frac{\exp[\beta_0 + \beta_1 x_1 \ldots + \beta_d x_d]}{1+\exp[\beta_0 + \beta_1 x_1 \ldots + \beta_d x_d]}$$

The 'logit' link is common, for a couple of good reasons.

One big reason: A little algebra shows

$$\log\left[\frac{p}{1-p}\right] = \beta_0 + \beta_1 x_1 \ldots + \beta_d x_d,$$

so that it is a linear model for log-odds.

Estimate the Probability of Default

Credit Card Clients does binary regression: Default versus not Default.
Say $y = 1$ for Default, otherwise $y = 0$.

`Default.csv` has for 30,000 credit card clients (about 6,636 default) related info.

Logistic regression fits $\mathrm{p}(y = 1)$ as a function of related information.

The following link gives more details on this data set.
Default of Credit Card Clients Data Set

Logistic regression is easy in Python

Again using glm:

```
smf.glm(formula=my_formula, data=default,
  family=sm.families.Binomial()).fit()
```

The argument '`family=sm.families.Binomial()`' indicates y is binary.

The reponse can take a few forms:

- y = 1, 1, 0, ... numeric vector.
- y = TRUE, TRUE, FALSE, ... logical.
- y = 'win','win','lose', ... factor.

Everything else is the same as for linear regression.

Interpreting Coefficients

The model is

$$\frac{p}{1-p} = \exp\left[\beta_0 + x_1\beta_1 + \ldots + x_p\beta_p\right]$$

So $\exp(\beta_j)$ is the odds multiplier for a unit increase in x_j.

`b['AGE']=0.007`, so one unit increase in `AGE` multiplies odds of default by $\exp(0.007) \approx 1.007$.

`b['PAY_0']=0.577`, so one unit increase in last month payment multiplies odds of default by $\exp(0.577) \approx 1.781$.

What is the odds multiplier for a covariate coefficient of zero?

The summary function gives coefficients, plus some other info.

```
proba.summary()
(Dispersion parameter for binomial family taken to be 1)
Residual deviance: 27877  on 29975  degrees of freedom
```

The same stuff is in output for our *linear* OJ regression.

```
reg_full.summary()
(Dispersion parameter for gaussian family taken to be 0.48)
Residual deviance: 13975  on 28935  degrees of freedom
```

These are stats on fit, and they are important in either linear or logistic regression. Understanding deviance ties it all together.

Estimation and Goodness of Fit (GOF)

Two related concepts:

Likelihood is a function of the unknown parameters β of a statistical model, given data:

$$L(\beta|data) = P(data|\beta)$$

Maximum Likelihood (ML) Estimation:

$$\hat{\beta} = \max_{\beta} L(\beta|data)$$

ML estimates $\hat{\beta}$ are those parameter values β that are most likely to have generated our data.

Estimation and Goodness of Fit (GOF)

Two related concepts:

Deviance refers to the distance between data and fit.
You want to make it as small as possible.

$$Dev(\beta) = \text{-2log } L(\beta|data) + \text{C}$$

C is a constant you can mostly ignore.

Deviance is useful for comparing models.

Deviance is a measure of GOF that plays the role of residual sums of squares for a broader class of models (logistic regression etc.)

We'll think of deviance as a cost to be minimized.

Minimize deviance $\Leftrightarrow$ maximize likelihood.

Least-Squares and deviance in linear regression

The probability model is $y \sim \mathrm{N}(\mathbf{x}'\boldsymbol{\beta}, \sigma^2)$.

density of $N(\mu, \sigma^2) = \exp\left[-(y-\mu)^2/2\sigma^2\right]/\sqrt{2\pi\sigma^2}$.

Given n independent observations, the likelihood is

$$\prod_{i=1}^{n} \mathrm{pr}(y_i|\mathbf{x}_i) \propto \exp\left[-\frac{1}{2}\sum_{i=1}^{n}\left(y_i - \mathbf{x}_i'\boldsymbol{\beta}\right)^2/\sigma^2\right]$$

This leads to Dev $\propto \dfrac{1}{\sigma^2}\displaystyle\sum_{i=1}^{n}\left(y_i - \mathbf{x}_i'\boldsymbol{\beta}\right)^2$.

Minimizing deviance is the same as least squares!

And thus the MLE minimizes our sum of squared errors.

MLE for Logistic Regression

Our logistic regression likelihood is the product

$$
\begin{aligned}
L(\beta) &= \prod_{i=1}^{n} P(y_i|\mathbf{x}_i) = \prod_{i=1}^{n} p_i^{y_i}(1-p_i)^{1-y_i} \\
&= \prod_{i=1}^{n} \left(\frac{\exp[\mathbf{x}_i'\boldsymbol{\beta}]}{1+\exp[\mathbf{x}_i'\boldsymbol{\beta}]}\right)^{y_i} \left(\frac{1}{1+\exp[\mathbf{x}_i'\boldsymbol{\beta}]}\right)^{1-y_i}
\end{aligned}
$$

This is maximized by minimizing the deviance

$$
\begin{aligned}
Dev &= -2\sum_{i=1}^{n} \left(y_i \log(p_i) + (1-y_i)\log(1-p_i)\right) \\
&\propto \sum_{i=1}^{n} \left[\log\left(1+e^{\mathbf{x}_i'\boldsymbol{\beta}}\right) - y_i\mathbf{x}_i'\boldsymbol{\beta}\right]
\end{aligned}
$$

We have the same output as for a linear/gaussian model.

But the '`dispersion parameter`' here is always set to one.
Check this to make sure you've actually run logistic regression.

```
proba.summary()
(Dispersion parameter for binomial family taken to be 1)
Residual deviance: 27877  on 29975  degrees of freedom
```

'degrees of freedom' is actually 'number of observations - df', where df is the number of coefficients estimated in the model.

That is, df(deviance) = nobs - df(regression).

From the Python output, how many observations do we have?

Sum of Squares (Deviance) is the bit we need to minimize:

$$Dev \propto \sum_{i=1}^{n} (y_i - \mathbf{x}_i \boldsymbol{\beta})^2$$

This makes the observed data *as likely as possible*.

Error variance σ^2 measures variability around the mean.
i.e., $\sigma^2 = \mathrm{var}(\varepsilon)$, where $\varepsilon_i = y_i - \mathbf{x}_i \boldsymbol{\beta}$ are the 'residuals'.

$$\hat{\sigma}^2 = \frac{1}{n-p-1} \sum \hat{\epsilon}_i^2$$

Python estimates σ^2 and calls it the `Scale`.
Even if we know $\boldsymbol{\beta}$, we only predict log sales *with uncertainty*.
e.g., there's a 95% probability of log sales in $\mathbf{x}'\boldsymbol{\beta} \pm 2\sqrt{0.48}$

Residual deviance D is what we've minimized, using $\mathbf{x}'\boldsymbol{\beta}$.

Null deviance D_0 is for the model where you don't use $\mathbf{x}$.
i.e., if you use $\hat{y}_i = \bar{y}$:

- $D_0 = \sum(y_i - \bar{y})^2$ in linear regression.
- $D_0 = -2\sum\left[y_i \log(\bar{y}) + (1 - y_i)\log(1 - \bar{y})\right]$ in logistic reg.

The difference between D and D_0 is due to info in x.
Proportion of deviance explained by **x** is called R^2:

$$R^2 = \frac{D_0 - D}{D_0} = 1 - \frac{D}{D_0}.$$

This measures how much variablity you are able to model.

in `proba`: $R^2 = 1 - 27877/31705 = 0.12$
in `reg_full`: $R^2 = 1 - 13975/30079 = 0.54$

R^2 in linear regression

These forumulas should look pretty familiar.

$R^2 = 1-$ SSE/SST from previous classes, and linear deviance is just the Sum of Squares!

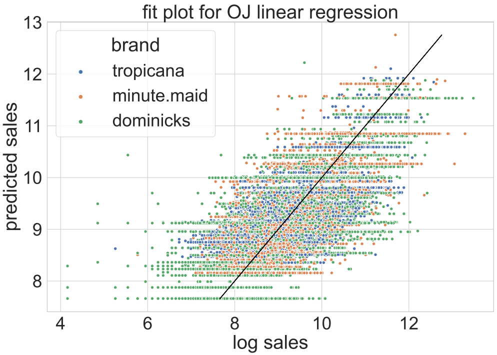
You'll also recall that $R^2 = \mathrm{cor}(y, \hat{y})^2$ in linear regression, where $\hat{y}$ denotes 'fitted value' $\hat{y} = f(\mathbf{x}'\hat{\boldsymbol{\beta}}) = \mathbf{x}'\hat{\beta}$ in lin reg.

```
np.corrcoef(fitted,log_sales)[0][1]**2
0.53539385
```

For linear regression, min deviance $=$ max $\mathrm{cor}(y, \hat{y})$.
If y vs $\hat{y}$ makes a straight line, you have a perfect fit.

R^2 increases after adding any explanatory variable.

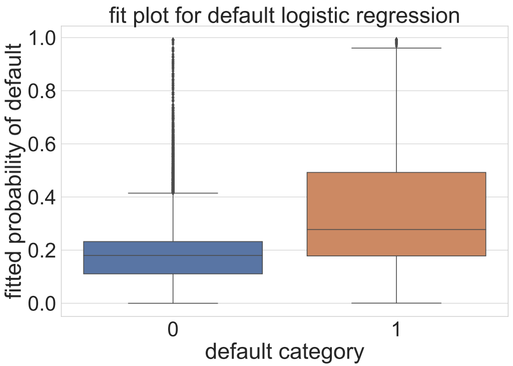
Fit plots: $\hat{y}$ vs y



It's good practice to plot $\hat{y}$ vs y as a check for misspecification. (e.g., non-constant variance, nonlinearity in residuals, ...)

Fit plots for logistic regression

We can plot $\hat{y}$ vs y in logistic regression using a boxplot.



The estimation pushes each distribution away from the middle.

Where would you choose for a classification cut-off?

Implementation

Linear regression has a closed-form (analytical) solution. $\hat{\beta} = (X'X)^{-1}X'y$.

Logistic regression requires a numerical procedure. For instance, we can adopt gradient descent of the loss function.
Gradient descent: applicable in the search of local minimum of a differentiable function. Extremely useful (with some twists) in deep learning.
Algorithm:

1. Start with an initial value β_0.
2. Iterate
$$\beta_{t+1} = \beta_t - \gamma_t \nabla loss(\beta_t).$$
3. Stop when converged.

γ_t is called learning rate – the most important parameter to tune.

Prediction

In logistic regression, we use `reg.predict(mynewdata)` to get probabilities $e^{\mathbf{x}'\hat{\boldsymbol{\beta}}}/(1+e^{\mathbf{x}'\hat{\boldsymbol{\beta}}})$:

```
proba.predict(default[0:4])
        1         2         3         4
 0.510777  0.152661  0.203579  0.246400
```

`newdata` *must* match the format of original data.

Out-of-Sample Prediction

You care about how your model predicts out-of-sample (OOS).

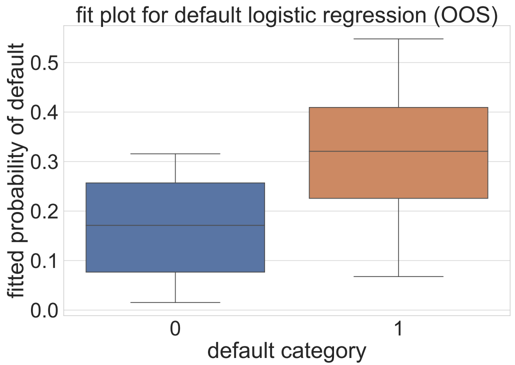
One way to test this is to use a validation sample.
Fit your model to the remaining *training data*,
and see how well it predicts the *left-out data*.

```
# Sample 1000 random indices
leaveout = sample(range(len(default)),1000)
# train the model WITHOUT these observations
probatrain = smf.glm(formula=my_formula ,
    data=default.drop(leaveout), family=sm.families.Binomial()).fit()

# predicted probability of default on the left out data
pdefault = probatrain.predict(default.iloc[leaveout])
```

Out-of-Sample Prediction

Fit plots on the 1000 left out observations.



For the left-out data, we can also evaluate deviances and R^2.

Since the sample is random, you might get different results.