Logistic Regression

classification

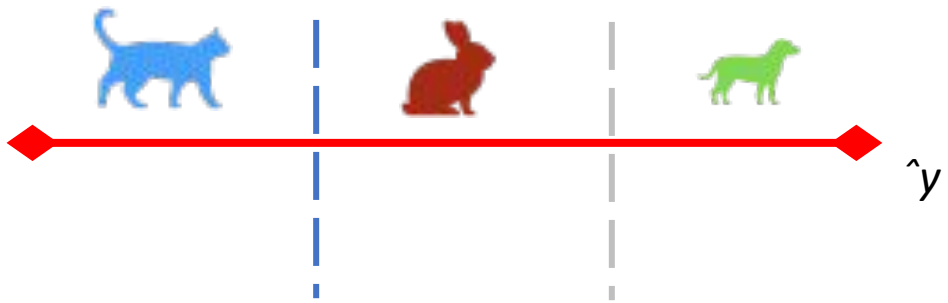
Régression logistique – classification - ML

- Classification ≠ Regression
- Logistic Regression
  - function logit
- Confusion Matrix
- AUC – F1 score
- logistic regression with scikit-learn
- categorical predictors
  - dummy encoding
  - binary encoding
- Imbalanced Datasets
- Multiclass Classification
  - one vs rest
  - one vs one

Binary classification : target variable is categorical

The output of linear regression is continuous

If we use linear regression to classify animals (cat = 0, rabbit = 1, dog = 2) we need arbitrary thresholds that do not reflect reality



Why would we have an arbitrary order between animals ?

does

- cats < rabbits < dogs

make more sense than

- Dogs < cats < rabbits

Furthermore, the output range of linear regression is not constrained or capped..

Can't use linear regression for classification

Logistic regression

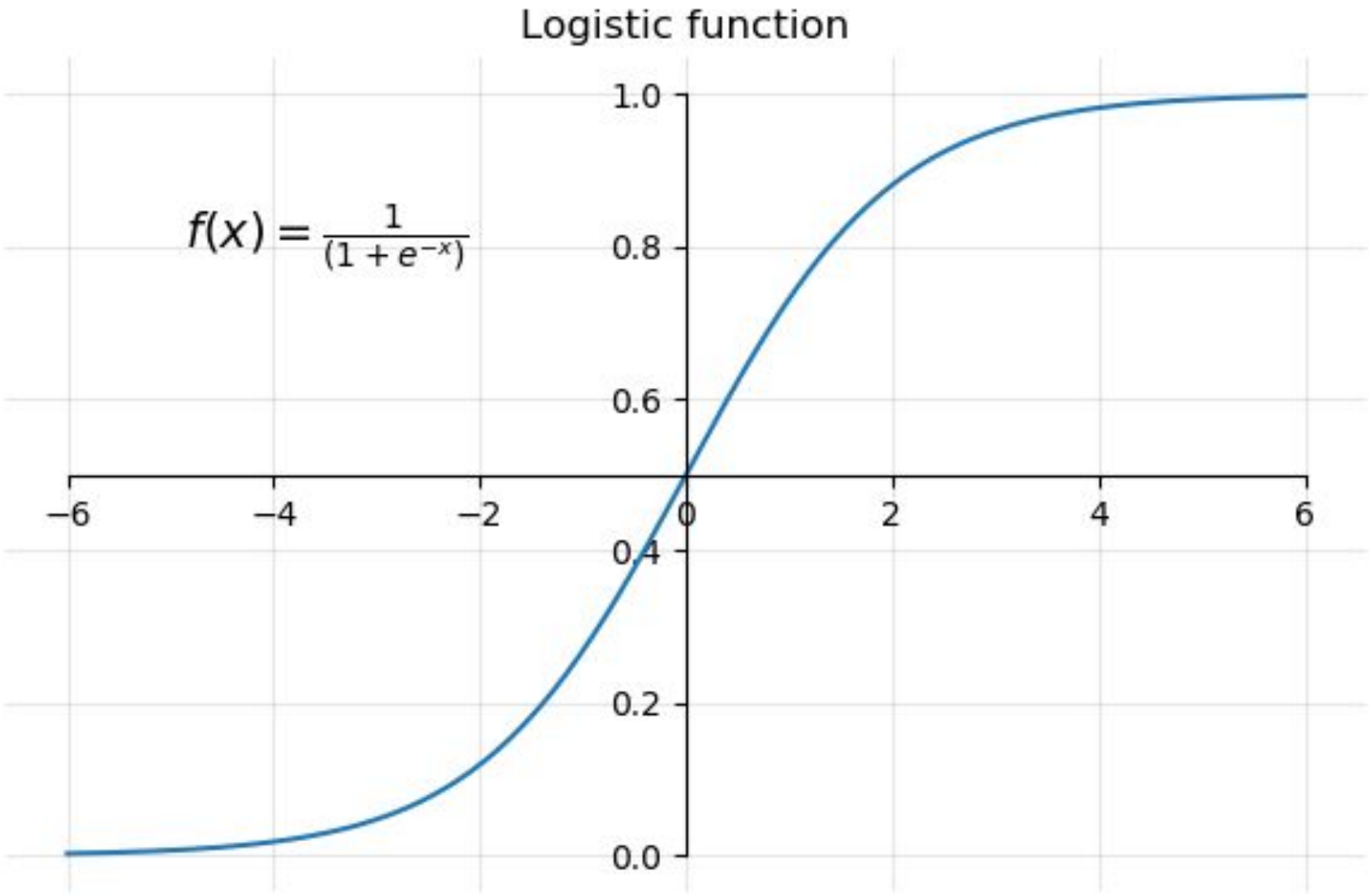
The main idea behind binary classification with logistic regression

Goal: Binary Classification: 0 / 1

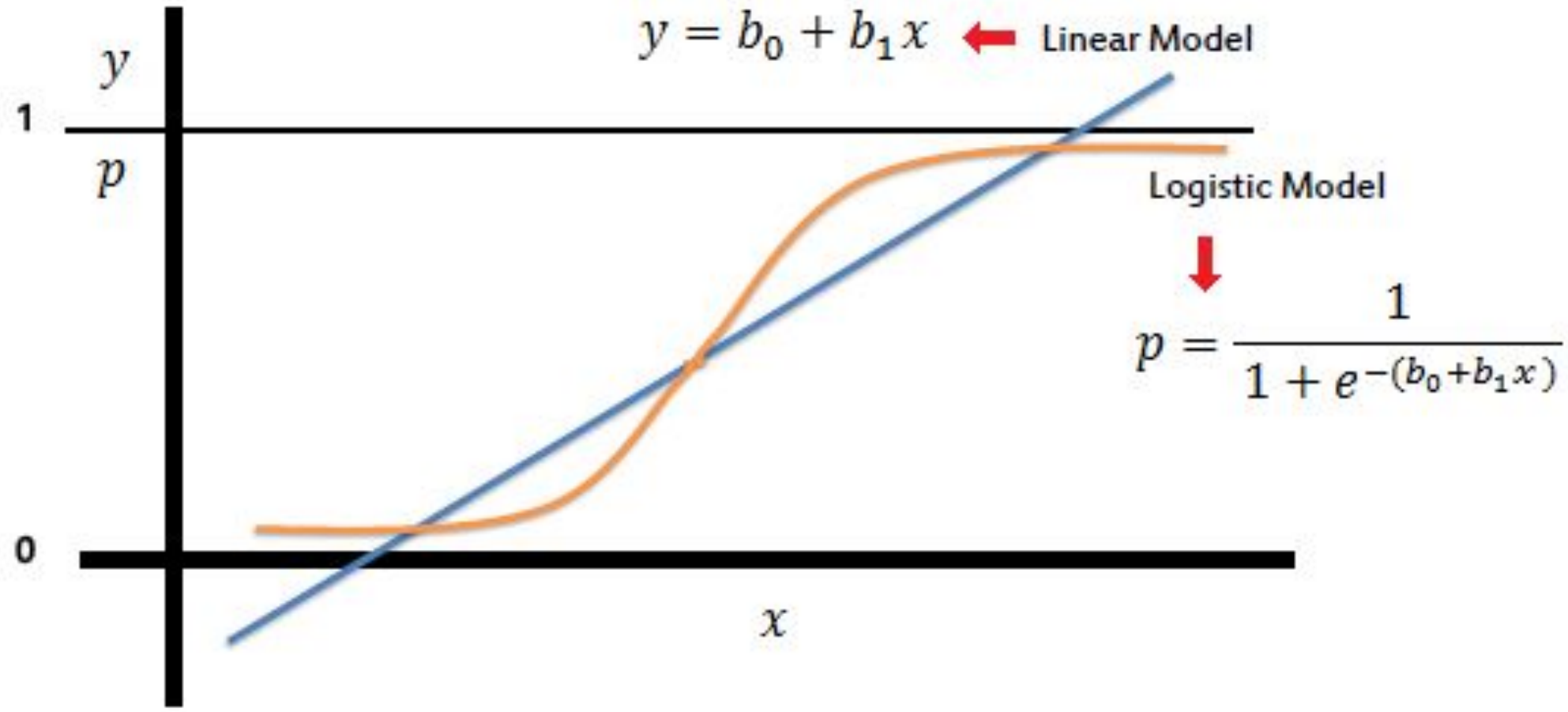
1. Use linear regression
2. Constrain the estimated values **Y** between the interval [0, 1]
3. Interpret the results as a probability **P** of belonging to one of the categories (0 or 1)
4. Define a threshold Tau = 0.5
5. Classify with the rule:
   - if **P** < Tau => Y belongs to category 0
   - else **P** > Tau => Y belongs to category 1

Logistic function

de R -> [0,1]



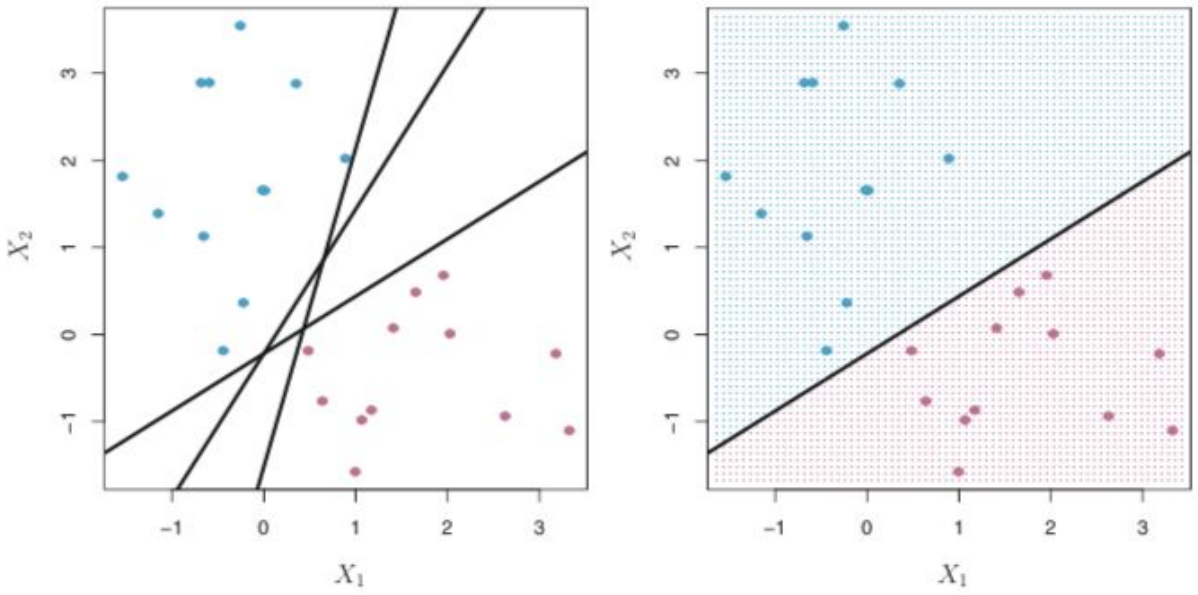
Logistique regression vs Linear regression



Find the best hyperplan that separates the data

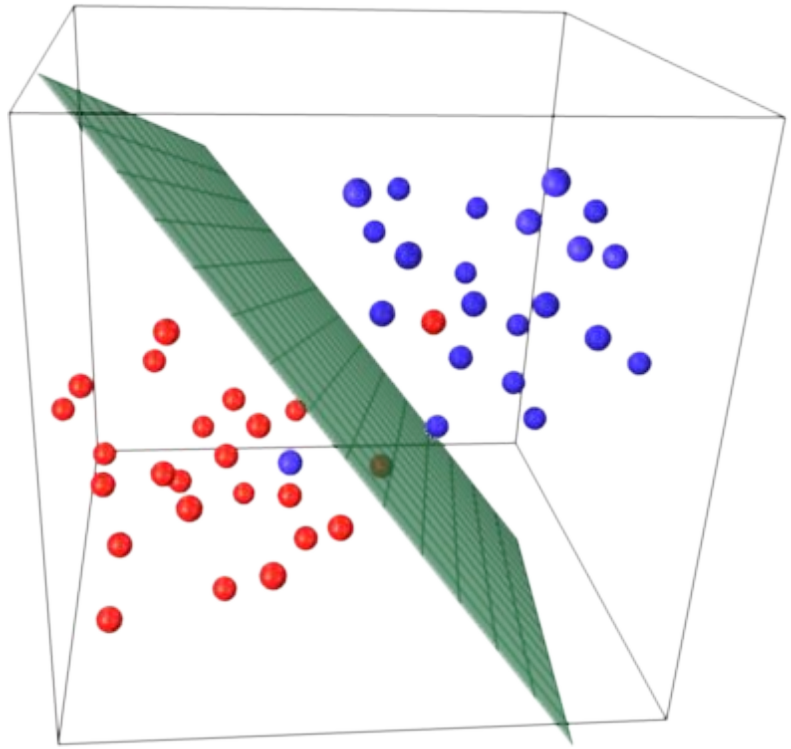
2D

$\hat{y} = a_1 x_1 + a_0$



3D

$\hat{y} = a_1 x_1 + a_2 x_2 + a_0$



Credit default dataset

Predictors:

- continuous: *balance*, *income*
- binary: *student*

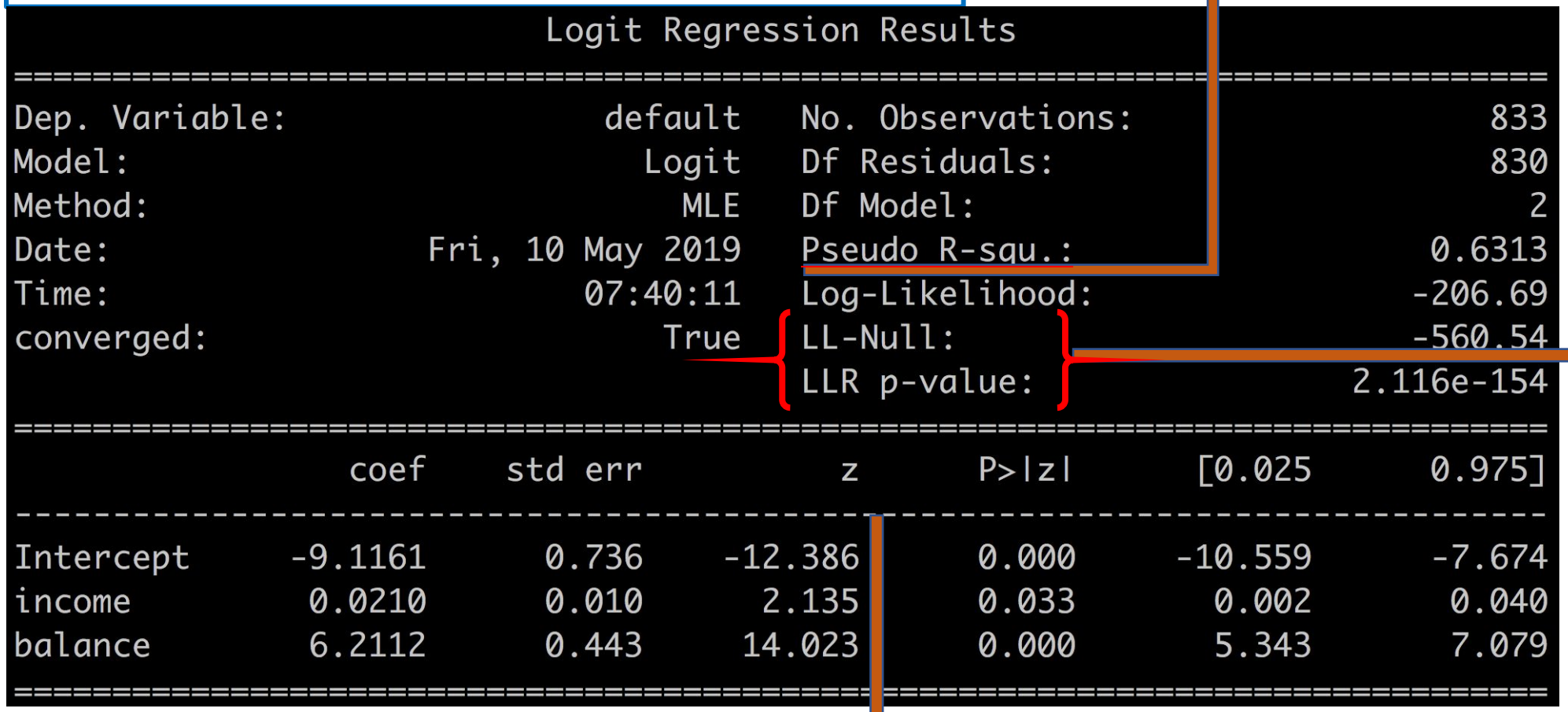
Target Variable:

- *default* indicates that a person defaulted on his/her payments
- *default* is a binary variable that takes values 0 or 1
  - No (0) 500
  - Yes (1) 333

```
import statsmodels.formula.api as smf
import pandas as pd

# Load the dataset
df = pd.read_csv(credit_default_sampled.csv')

# instantiate the model le modèle
model = smf.logit('default ~ income + balance', data = df)

# Fit the model to the data
results = model.fit()

# Results
results.summary()
```

Logistic regression result

default ~ income + balance



Logit Regression Results

Dep. Variable:	default	No. Observations:	833
Model:	Logit	Df Residuals:	830
Method:	MLE	Df Model:	2
Date:	Fri, 10 May 2019	Pseudo R-squ.:	0.6313
Time:	07:40:11	Log-Likelihood:	-206.69
converged:	True	LL-Null:	-560.54
		LLR p-value:	2.116e-154

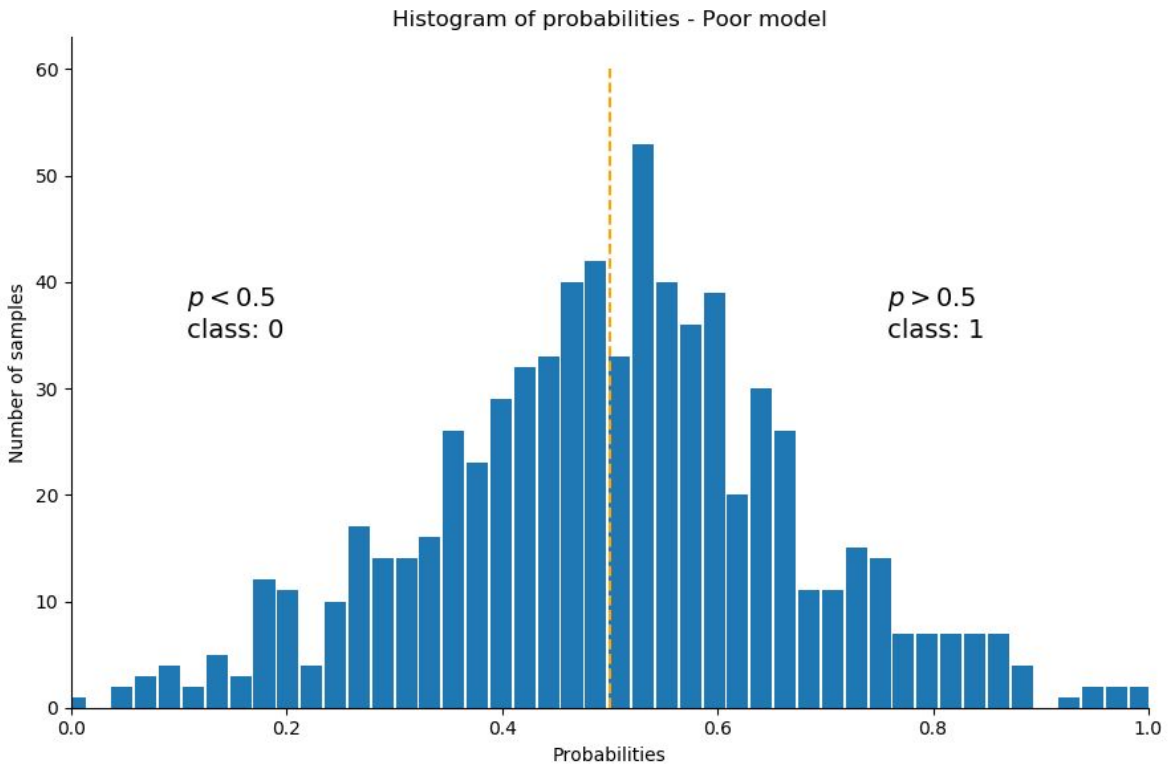
	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-9.1161	0.736	-12.386	0.000	-10.559	-7.674
income	0.0210	0.010	2.135	0.033	0.002	0.040
balance	6.2112	0.443	14.023	0.000	5.343	7.079

Probabilities histogram

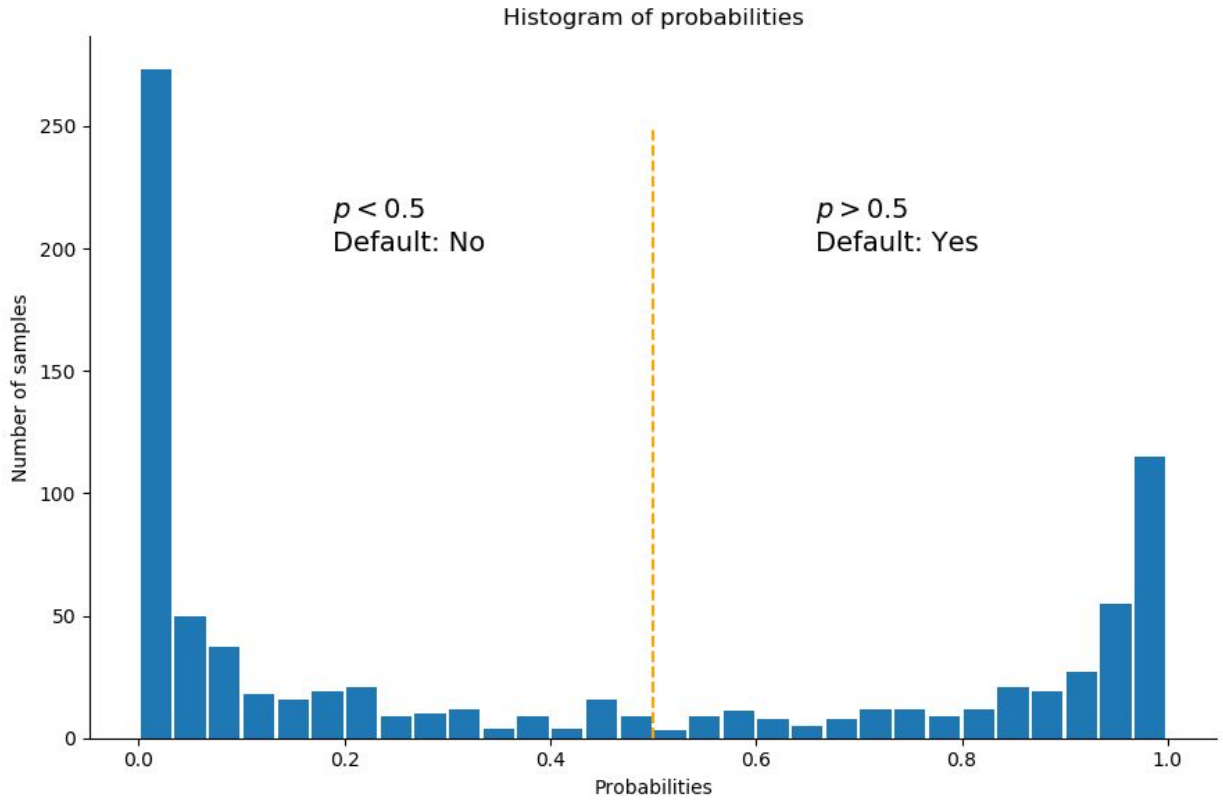
The histogram of estimated values provides a good indication of the model's separation power.

```
y_proba = results.predict(df[['income', 'balance']])
```

Poor model
very confused



Excellent model
clear separation



Classes prédites en fct des probabilités de prédiction

```
# output of the model predictions as probabilities, yhat in [0,1]

y_proba = results.predict(df[['income', 'balance']])


# transform the probabilities into a class

predicted_class = (y_proba > 0.5).astype(int)

print(predicted_class)
> [1,1,1,1...,0,0,0]
```

Note that the choice of the classification threshold (0.5) remains arbitrary.

confusion matrix

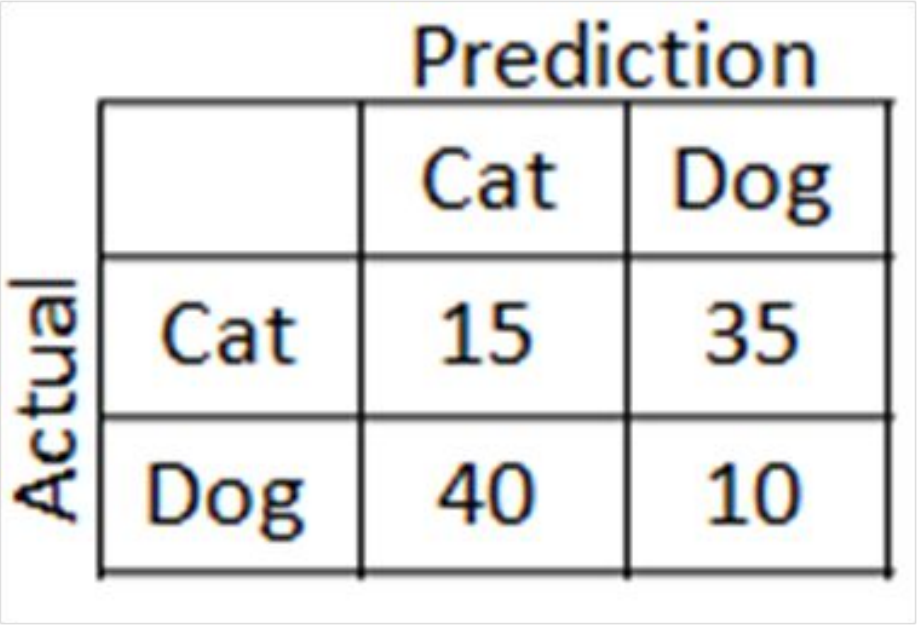
4 possible cases

2 correct ones:
- 1 classified as 1
- 0 classified as 0

2 false ones:
- 1 classified as 0
- 0 classified as 1

```
results.pred_table()
```

	Predicted 1	Predicted 0
Actual 1	True Positives	False Negatives
Actual 0	False Positives	True Negatives



Confusion matrix: default ~ income + balance

	Predicted Default	Predicted Non-Default
Actual Default	286	47
Actual Non-Default	40	460

Out of of 333 *default* samples:

- 286 were **correctly** predicted as *default* by our model (True positives)
- 47 were **wrongly** predicted as *non-default* by our model (False Negatives)

And out of 500 *non-default* samples,

- 460 were **correctly** predicted as *non-default* by our model (True Negatives)
- 40 were **wrongly** predicted as *default* by our model (False Positives)

classification metrics

	Predicted 1	Predicted 0
Actual 1	True Positives	False Negatives
Actual 0	False Positives	True Negatives

We can define multiple metrics using the confusion matrix

- Accuracy = (TP + TN) / (P + N)
- True Positive Rate = (TP) / (TP + FP)
- False Positive Rate = (FP) / (TP + FP)

	Predicted Default	Predicted Non-Default
Actual Default	286	47
Actual Non-Default	40	460

- number of samples correctly classified : 460 + 286 = 746
- total number of samples : 833
- so our **accuracy** is 746 / 833 **= 89.56%**

- **TPR = 286 / 333 = 85.89%**
- **FPR = 47 / 333 = 14.11%**

Confusion matrix - (the name is fitting)

		True condition			
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$
Predicted condition	Predicted condition positive	**True positive**, Power	**False positive**, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$
		True positive rate (TPR), Recall, Sensitivity, probability of detection = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$	F_1 score = $\frac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}$

from Wikipedia Confusion Matrix page

Your turn – default ~ income + balance + student

Build the model: default ~ income + balance + student

1. instantiate the `smf.logit` model
2. Fit the model
3. interpret the results
4. calculate the confusion matrix
5. calculate accuracy, TPR, FPR
6. plot the histogram of probabilities

What about the threshold ?

What happens if we use a different classification threshold ?

The confusion matrix and associated metrics also change.

t = 0.5	Predicted Default	Predicted Non-Default
Actual Default	286	47
Actual Non-Default	40	460

Acc (0.5) = 746 / 833 **= 89.56%**
TPR = 286 / 333 = 85.89%
FPR = 47 / 333 = 14.11%

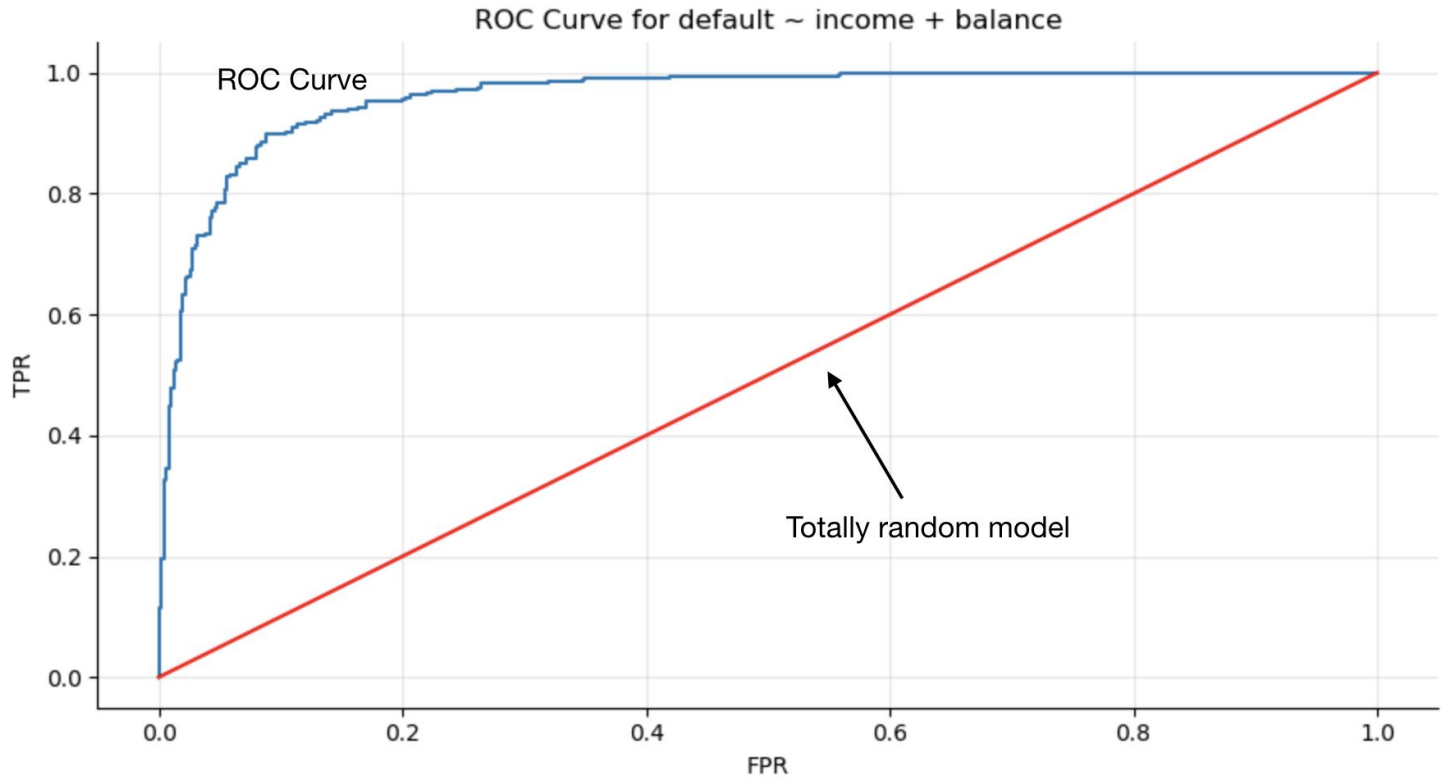
t = 0.75	Predicted Default	Predicted Non-Default
Actual Default	243	90
Actual Non-Default	19	481

Acc (0.75)=(243 + 481)/888 = 81 %
TPR = 243 / 333 = 72.97%
FPR = 90 / 333 = 27.03%

ROC-AUC

Plotting the TPR vs. FPR curve while varying the threshold from 0 to 1 gives the **ROC** curve.

The area under the curve (**AUC**) is a more robust metric than accuracy.



See here and here

```
yhat= results.predict(df)

from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(df['default'], yhat)

from sklearn.metrics import roc_auc_score
score = roc_auc_score(df['default'], yhat)
```

Your turn – default ~ income + balance + student

For the model **default ~ income + balance + student**:

- Build the model
- Interpret the results
- Compute the confusion matrix
- Calculate accuracy, TPR, and FPR

- Compute AUC
- Plot the ROC curve

Compare with the model **default ~ income**.

Imbalanced datasets

The accuracy paradox

The original **credit default** dataset contains:

- **333** default cases
- **9667** non-default cases

A (stupid, useless) model that always predicts **non-default** has an accuracy of **96.67%**.

The target class is a strong minority.
The dataset is **imbalanced**.

Four strategies to handle class imbalance:

- **Undersampling** the majority class
- **Oversampling** the minority class
- **Combination** of over- and undersampling
- **SMOTE**

Balance the classes

Stratégies pour résoudre le problème du déséquilibre de la classe minoritaire

- sous-échantilloner la classe majoritaire
- sur-échantilloner la classe minoritaire

On peut aussi

- mélanger les 2 approches: sur et sous-échantillonage

Le but n'est pas de balancer parfaitement les classes 50 / 50

A vous – imbalanced dataset

On the full **credit default** dataset:

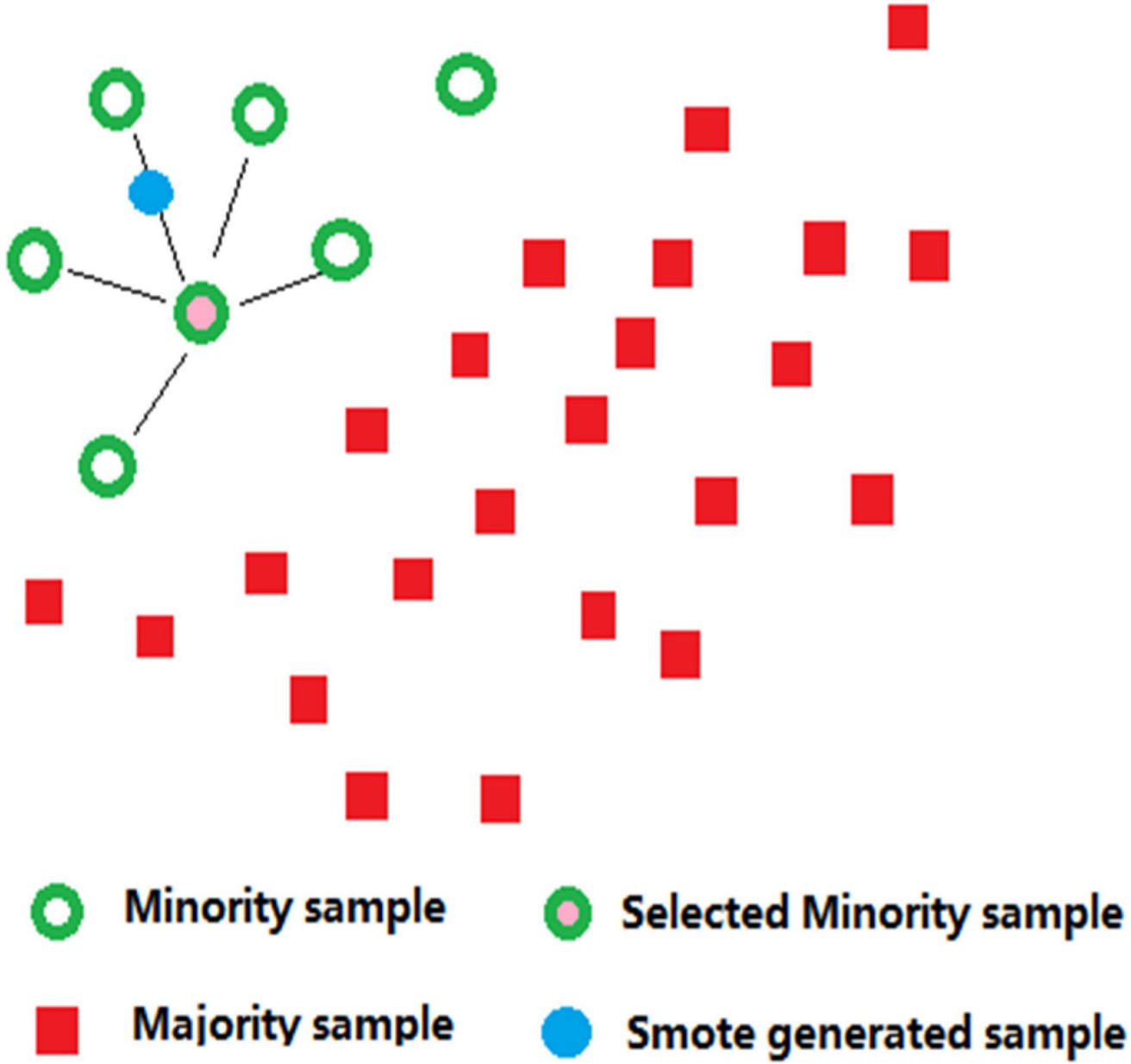
- Build the model **default ~ income + balance + student**.
- Compute **accuracy** and **AUC**, and plot the **ROC curve**.

Then:

- **Oversample** the minority class.
- **Undersample** the majority class.
- Vary the **ratios**.

```
# over sample the minority class
df = pd.read_csv('credit_default.csv')
minority = df[df.default == 0].sample(n = 2000, replace = True)
majority = df[df.default == 1]

data = pd.concat(minority, majority)
# shuffle
data = data.sample(frac = 1)
```

SMOTE



The SMOTE algorithm is implemented by randomly selecting one of the few classes of sample data, and then selecting several of its neighboring sample data by linear interpolation.

The SMOTE algorithm generates artificial samples in three steps,

1. selecting a few random sample.
2. Select instance from its K-nearest minority class neighbors.
3. Finally, a new sample is created by randomly interpolating two samples

from https://www.mdpi.com/2227-9717/10/7/1420

SMOTE does not work!

Imbalanced learn

A Python Package to Tackle the Curse of Imbalanced Datasets in Machine Learning http://imbalanced-learn.org

Tips for Handling Imbalanced Data in Machine Learning - MachineLearningMastery.com

Encoding Categorical variables

Encoding Categorical variables

One-hot encoding – Dummy encoding

How to convert categorical variables into numerical variables?

Binary:

- Yes/No
- 1/0
- Male/Female
- Spam/Legit
- Action: Buy, Save, Login

Multinomial, non-ordinal:

- List of cities, countries, destinations, age groups
- Education level
- Car brands

Examples

Car brand: Audi, Renault, Ford, Fiat

If an arbitrary number is assigned to each brand, a hierarchy is created:

- Audi → 1, Renault → 2, Ford → 3, Fiat → 4

Similarly:

- Dog, cat, mouse, chicken → {1,2,3,4}
- Why would a chicken be four times a dog?

Sometimes, assigning a number to each category makes sense—**ordered categories**:

- Child, young, adult, elderly → {1,2,3,4}
- Negative, neutral, positive → {-1, 0, 1}

Prédicteurs catégoriques – dummy encoding

Load auto-mpg

Origin (3) and name (many) are unordered (non-ordinal) categories.
How to include **origin** as a predictor in a linear model?

With **Pandas**, create one variable per category:

- **American:** 0 or 1
- **European:** 0 or 1
- **Japanese:** 0 or 1

`pd.get_dummies()` creates **N-1** variables.
Then, define the model:
mpg ~ American + European

```
origin_variables = pd.get_dummies(df.origin)
df = df.merge(origin_variables, left_index=True, right_index= True )
results = smf.ols('mpg ~ Japanese + European', data = df).fit()
```

N-1 new variables are needed for N categories.

```
# Essayer
results = smf.ols('mpg ~ Japanese + European + American', data = df).fit()
```

Prédicteurs catégoriques – statsmodel

Statsmodels encodes categorical variables directly with

```
mpg ~ C(origin)
```

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	20.0335	0.409	49.025	0.000	19.230	20.837
origin[T.European]	7.5695	0.877	8.634	0.000	5.846	9.293
origin[T.Japanese]	10.4172	0.828	12.588	0.000	8.790	12.044

Interprétation des coefficients - catégories

La moyenne par catégorie

```
df[['mpg','origin']].groupby(by = 'origin').mean().reset_index()
```

```
     origin        mpg
0  American  20.033469
1  European  27.602941
2  Japanese  30.450633
```

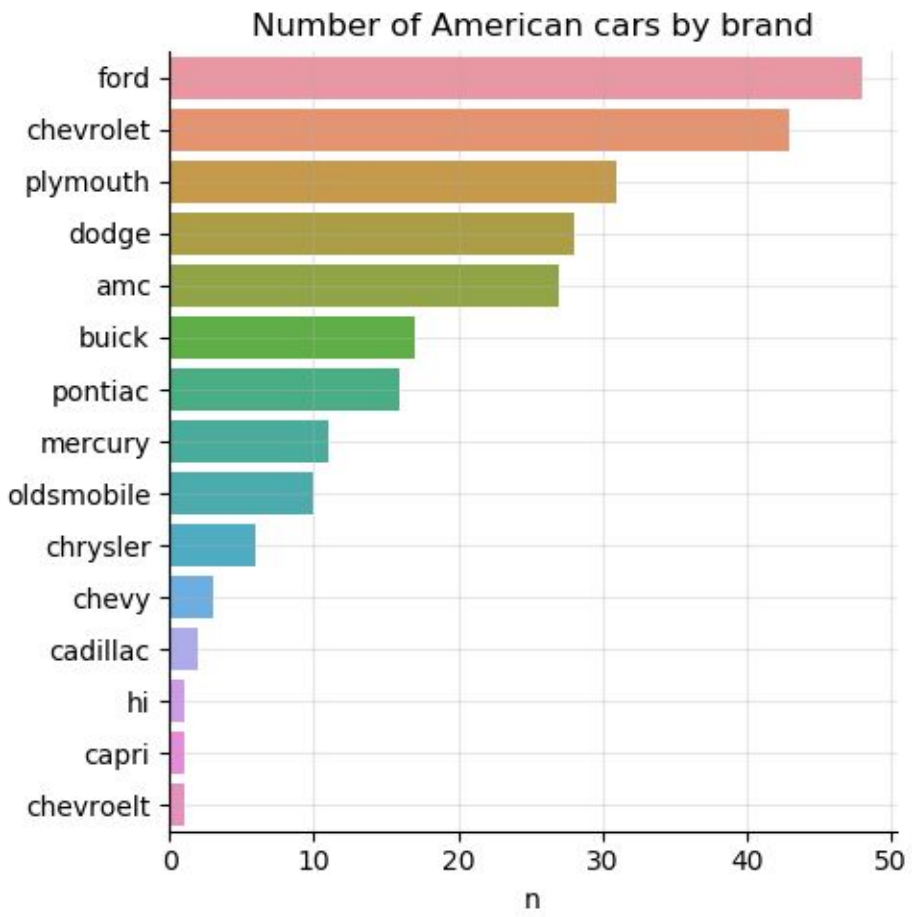
mpg ~ C(Origin)

```
Intercept           20.033469
origin[T.European]   7.569472
origin[T.Japanese]  10.417164
```

- Intercept = mpg_American
- origin[T.European] = mpg_European − mpg_American
- origin[T.Japanese] = mpg_Japanese − mpg_American

Prédicteurs catégoriques

How to include car brand in auto-mpg?
There are **36 categories**.
Some categories have **few samples**.



Binary encoding!

```
import category_encoders as ce
# define the encoder
encoder = ce.BinaryEncoder(cols=['brand'])
df = encoder.fit_transform(df)
```

Instead of

- mpg ~ brand

We use the model

- mpg ~ brand_0 + brand_1 + … + brand_5

Binary encoding

Temperature	Order	Binary
Hot	1	001
Cold	2	010
Very Hot	3	011
Warm	4	100
Hot	1	001
Warm	4	100
Warm	4	100
Hot	1	001
Hot	1	001
Cold	2	010

Temperature_0	Temperature_1	Temperature_2
0	0	1
0	1	0
0	1	1
1	0	0
0	0	1
1	0	0
1	0	0
0	0	1
0	0	1
0	1	0

https://towardsdatascience.com/all-about-categorical-variable-encoding-305f3361fd02/

Number of binary variables = INT(log2(Number of categories))

Category encoders - the library

https://contrib.scikit-learn.org/category_encoders/index.html

A set of scikit-learn-style transformers for encoding categorical variables into numeric with different techniques.

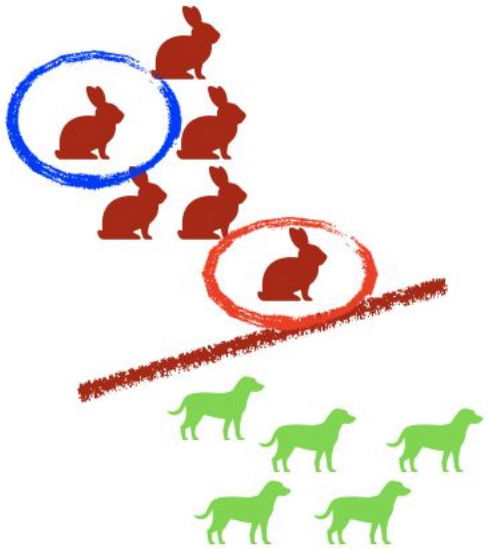
- Backward Difference Coding
- BaseN
- Binary
- CatBoost Encoder
- Count Encoder
- Generalized Linear Mixed Model Encoder
- Gray
- Hashing
- Helmert Coding
- James-Stein Encoder
- Leave One Out
- M-estimate
- One Hot
- Ordinal
- Polynomial Coding
- Quantile Encoder
- RankHotEncoder
- Sum Coding
- Summary Encoder
- Target Encoder
- Weight of Evidence
- Wrappers

Multiclass Classification - Multinomial

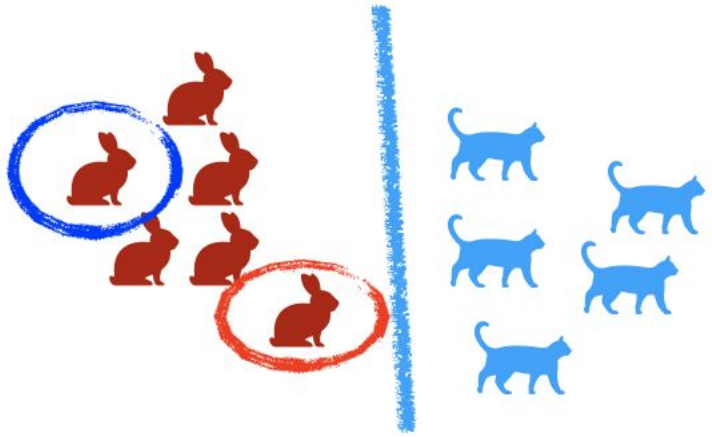
Classification multinomiale

One vs One

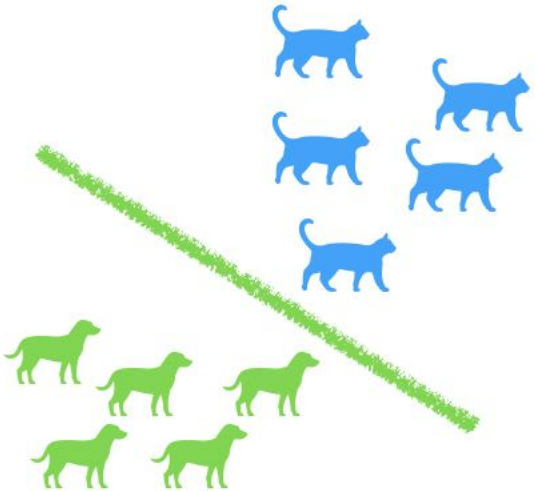
Pour N catégories on construit N modèles
Le modèle final est obtenu par vote ou par moyenne



M1: rabbits vs dogs



M2: cats vs rabbits

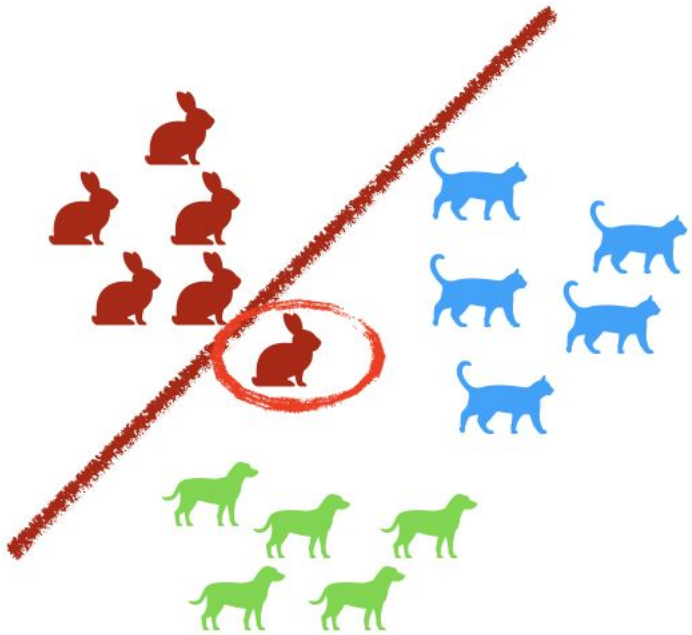


M3: dogs vs cats

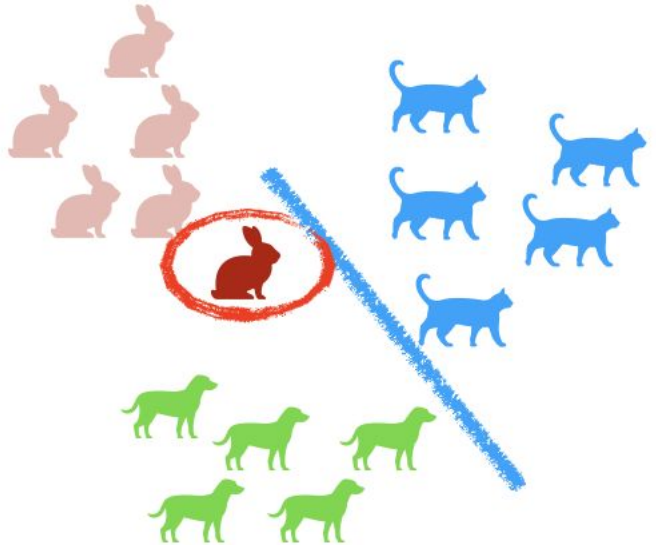
Classification multinomiale

One vs Rest

N-1 modèles sont nécessaires
propagation de l'erreur



M1: rabbits vs cats and dogs (non rabbits)



M2: cats vs dogs