# INTRODUCTION

# To Data Science

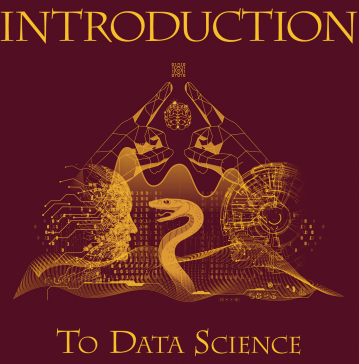
# Introduction to Data Science

## Linear and Logistic Regression - Class 9


## Giora Simchoni

gsimchoni@gmail.com and add #intro2ds in subject

## Stat. and OR Department, TAU

# Intro. to Predictive Modeling

INTRODUCTION

TO DATA SCIENCE

# Predictive modeling and Supervised learning

- Basic idea: each observation is made of a vector $x \in \mathcal{X}$ (for example $x \in \mathbb{R}^p$) and a scalar $y$
- Our goal is to build a model of the relationship between $x$ and $y$:

$$y \approx f(x)$$

- $x$: predictors, regressors, features, exogenous variables
- $y$: response, dependent variable, endogenous variable

# Predictive modeling and Supervised learning

Two distinct goals for this:

1. Prediction: in the future we will get $x$ and have to *predict* $\hat{y} = f(x)$
2. Inference/understanding/model selection: Understanding the nature of the dependence between $x$ and $y$:
    - Which variables in $x$ are important for explaining or predicting $y$?
    - What type of dependence does $y$ have on $x$: linear? more complex?

- Regression: $y \in \mathbb{R}$ numeric
- Classification: $y \in \mathcal{G}$ an unordered set

# Wikiart paintings: a classification problem

$x \in \mathbb{R}^{K \times K \times 3}$: the image itself

$y \in \{\text{impressionist}, \text{realist}\}$

- More involved example: Cifar-10 with 10 classes
- A good model: $f(x)$ such that $f(x) \approx 1$ for impressionist and $f(x) \approx 0$ for realist
- Possible $f$: threshold the average red value for all pixels
- Does not do a very good job in separating impressionist from realist paintings…

# Netflix movies: a regression problem (sort of)

- Recall we had $x \in \mathbb{R}^{99}$ movies, plus one special (Miss Congeniality) that we will call $y$
  - All $x$ values are not really in $\mathbb{R}$ but in $\{0 = \text{None}, 1, 2, 3, 4, 5\}$
  - $y$ is in $\{1, 2, 3, 4, 5\}$ (no missing)
- A good model $f(x)$ sees the scores a user gave to the 99 movies (including which are missing) and gives a value that is close to $y$ for the same user

# Some more examples from real life

## Genome-Wide Association Studies (GWAS): find genetic causes of disease

- $y \in \{\text{sick}, \text{healthy}\}$ for specific disease
- $x \in \{0, 1, 2\}^{1M}$ ($p = 10^6$) number of copies of "risk" variant in each location in the genome
- The goal is to understand which coordinates in $x$ are related to $y$, and predict risk of $y$ for new people

## Email spam detection:

- $y \in \{\text{OK}, \text{spam}\}$ for each email
- $x$ can include sender identity, words and terms ("prize!", "sex", ...)
- The model should identify and remove spam

# Some more examples from real life

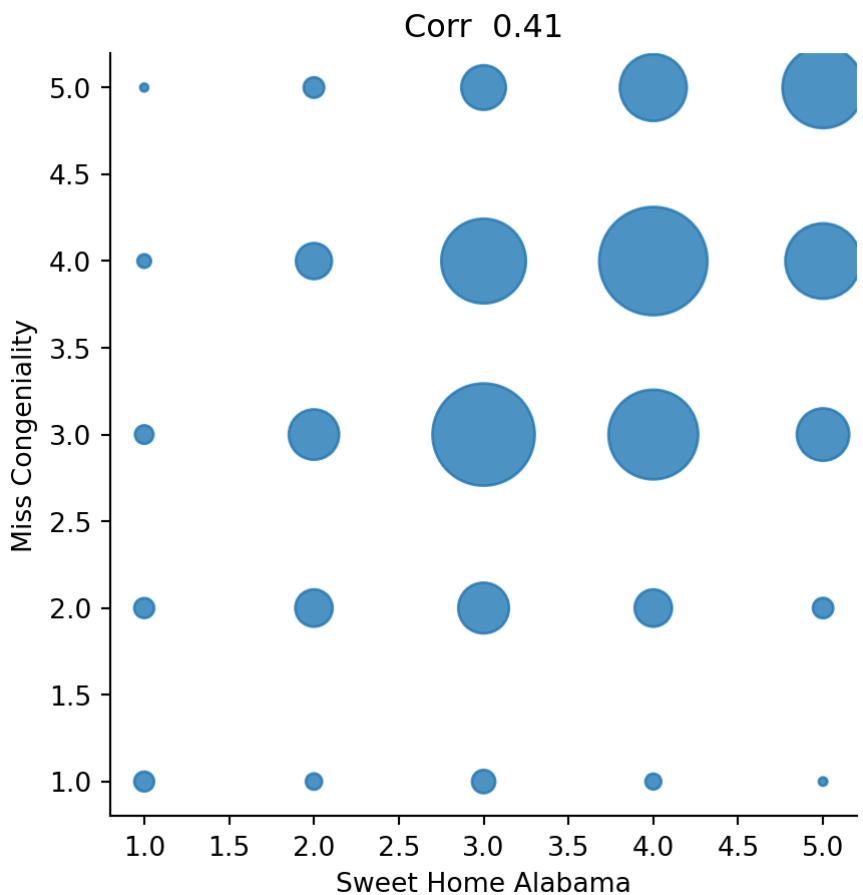
## Online advertising:

- Surfer arrives on website, need to decide if and what ad to show them
- $y$ can be the amount she will spend if shown advertising for shirt/shoes/car/home
- $x$: surfing history, location, time of day/week/year, information from other databases, …

# Some simple models for Netflix

The same score as a similar movie, say Sweet Home Alabama:

```
scatter_cong('Sweet Home Alabama')
```



# Some simple models for Netflix

The first PC score (those who love everything, love Miss Congeniality?):

- Code

# Predictive modeling paradigm

- We typically assume that we have a *training* dataset of size $n$:
  $Tr = \{(x_1, y_1), \ldots, (x_n, y_n)\} = (X_{n\times p}, Y_{n\times 1})$
- IID assumption: each pair $(x_i, y_i)$ is drawn indepednently from some distribution $P_{x,y}$
- A modeling approach takes $Tr$ as input and outputs a *prediction model* $\hat{f}(x)$ based on the training data
  - In prediction: we get a new value $x_0$ and predict $\hat{y}_0 = \hat{f}(x_0)$.
- How good is our prediction? We typically define a loss function $L(y, \hat{y})$ and the quality of the model is $\mathbb{E}_{x_0,y_0}(L(y_0, \hat{y}_0))$

# The loss function $L$

- It measures the quality of the prediction: we can think of $L(y, \hat{y})$ as a measure of how much we lose when we predict $\hat{y}$ but the truth is $y$.
- Simple example for classification: *misclassification error loss*

$$L(y, \hat{y}) = \begin{cases} 0 & \text{if } y = \hat{y} \\ 1 & \text{if } y \neq \hat{y} \end{cases}$$

- More complex approach: penalize different types of error differently, e.g.:

$$L(y, \hat{y}) = \begin{cases} 0 & \text{if } y = \hat{y} \\ 1 & \text{if } y = 0, \hat{y} = 1 \\ 10 & \text{if } y = 1, \hat{y} = 0 \end{cases}$$

- Simple example for regression: *squared error loss*

$$L(y, \hat{y}) = (y - \hat{y})^2.$$

Intro to Data Science

# Evaluating predictive models

- We are interested in $\mathbb{E}_{x_0,y_0}\left(L(y_0, \hat{y}_0)\right)$, but we don't know it
- Solution: in addition to the training data $Tr$, have a *test* data $Te = \{(x_{n+1}, y_{n+1}), \ldots, (x_{n+m}, y_{n+m})\}$ of size $m$ and evaluate the model on it: $\hat{Err} = \frac{1}{m}\sum_{i=n+1}^{n+m} L(y_i, \hat{f}\left(x_i\right))$.
- For squared error loss, it is typical to report the *Root* mean squared error:

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=n+1}^{n+m}\left(y_i - \hat{f}\left(x_i\right)\right)^2}$$

- Since we typically only have one dataset (as in Netflix, wikiart examples), we split it *randomly* in two parts:
  - Training set (typically $80\%$ of the data)
  - Test set (typically $20\%$ of the data)

# Data Splitting

Let's divide our Netflix data 80-20:

```
 1  X = ratings.values
 2  Y = miss_cong.values[:, 0]
 3  n = X.shape[0]
 4  tr_size = int(0.8 * n)
 5  te_size = n - tr_size
 6  tr_ind = np.random.choice(range(n), tr_size, replace=False)
 7  Xtr = X[tr_ind,]
 8  Xte = np.delete(X, tr_ind, axis=0)
 9  Ytr = Y[tr_ind]
10  Yte = np.delete(Y, tr_ind)
11
12  print(f'No. of train rows: {Xtr.shape[0]}, no. train of cols: {Xtr.shape[1]}')
13  print(f'No. of test rows: {Xte.shape[0]}, no. test of cols: {Xte.shape[1]}')
14  print(f'no. of obs in train y: {Ytr.shape[0]}')
15  print(f'no. of obs in test y: {Yte.shape[0]}')
```

```
No. of train rows: 8000, no. train of cols: 99
No. of test rows: 2000, no. test of cols: 99
no. of obs in train y: 8000
no. of obs in test y: 2000
```

# Linear Regression

INTRODUCTION

TO DATA SCIENCE

# Linear Regression

- Assume now $x \in \mathbb{R}^p$, $y \in \mathbb{R}$, and we want to build a model of the form:

$$\hat{f}(x) = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \ldots + \hat{\beta}_p x_p.$$

- We have $Tr$, how can we estimate the coefficients?
- Find coefficients that "fit" $Tr$ well, that is $\hat{f}(x_i) \approx y_i,\ i = 1, \ldots, n.$
- Possible approach: Minmize *residual sum of squares* (RSS):

$$RSS(\beta_0, \beta_1, \ldots, \beta_p) = \sum_{i=1}^{n} (y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \ldots + \hat{\beta}_p x_{ip}))^2 = \|Y - X_{n\times(p+1)}\beta|$$

- This is the *ordinary least squares (OLS) linear regression* problem

# Simple Demo: $p = 1$ on Netflix Data

Let's go back to $y$ = Miss Congeniality vs. $x_1$ = Sweet Home Alabama:

The `statsmodels` approach:

```
1  sweet_home_idx = 9
2  X_sweet_tr = Xtr[:, [sweet_home_idx]]
3
4  import statsmodels.api as sm
5
6  X_sweet_tr1 = sm.add_constant(X_sweet_tr)
7  model = sm.OLS(Ytr, X_sweet_tr1)
8  model = model.fit()
9  print(f'y = {model.params[0]:.2f} + {model.params[1]:.2f}*x1')
```

```
y = 2.13 + 0.40*x1
```

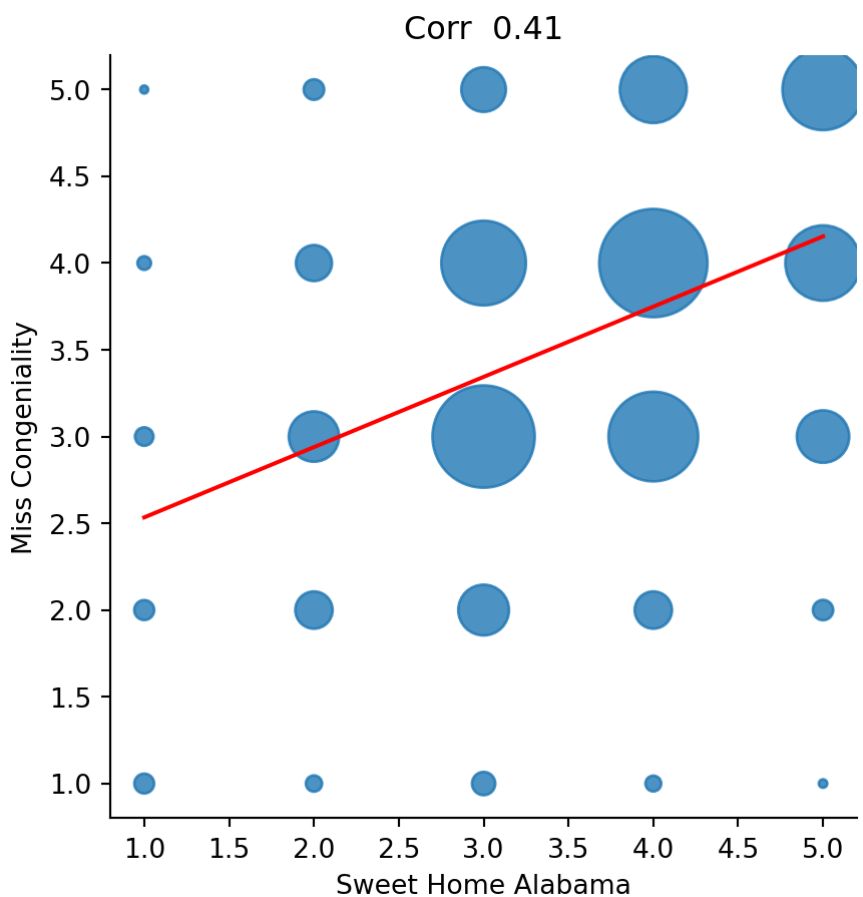
The `SKlearn` approach:

```
1  from sklearn.linear_model import LinearRegression
2
3  model = LinearRegression()
4  model.fit(X_sweet_tr, Ytr)
5  print(f'y = {model.intercept_:.2f} + {model.coef_[0]:.2f}*x1')
```

```
y = 2.13 + 0.40*x1
```

# The model is a simple straight line:

```
1 pred_x = np.arange(1, 6).reshape((5, 1))
2 y_hat = model.predict(pred_x)
3
4 scatter_cong('Sweet Home Alabama')
5 plt.plot(pred_x, y_hat, color = 'r')
6 plt.show()
```



# Evaluating on the test data

```
# this is the sklearn approach, no need to add constant
X_sweet_te = Xte[:, [sweet_home_idx]]
y_hat_te = model.predict(X_sweet_te)

test_RMSE_null = np.sqrt(np.mean((Yte-np.mean(Ytr))**2))
test_RMSE_1movie = np.sqrt(np.mean((Yte-y_hat_te)**2))

print(f'Test RMSE predicting the mean: {test_RMSE_null: .2f}')
print(f'Test RMSE with Sweet Home Alabama: {test_RMSE_1movie: .2f}')
```

```
Test RMSE predicting the mean:  0.97
Test RMSE with Sweet Home Alabama:  0.88
```

# Simple Demo: $p = 14$ on Netflix Data

```
Xtr_df = pd.DataFrame(Xtr[:, :14], columns=movies['title'][:14])
Xtr_df1 = sm.add_constant(Xtr_df)

model = sm.OLS(Ytr, Xtr_df1)
model = model.fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.276
Model:                            OLS   Adj. R-squared:                  0.274
Method:                 Least Squares   F-statistic:                     217.0
Date:                Mon, 14 Aug 2023   Prob (F-statistic):               0.00
Time:                        11:02:03   Log-Likelihood:                -9694.0
No. Observations:                8000   AIC:                         1.942e+04
Df Residuals:                    7985   BIC:                         1.952e+04
Df Model:                          14
Covariance Type:            nonrobust
================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const                        0.4074      0.090      4.509      0.000       0.230       0.584
Independence Day             0.0668      0.013      4.996      0.000       0.041       0.093
The Patriot                 -0.0161      0.012     -1.382      0.167      -0.039       0.007
The Day After Tomorrow       0.0503      0.011      4.612      0.000       0.029       0.072
Pirates of the Caribbean     0.0642      0.012      5.375      0.000       0.041       0.088
Pretty Woman                 0.1522      0.012     13.135      0.000       0.129       0.175
Forrest Gump                -0.0673      0.014     -4.953      0.000      -0.094      -0.041
The Green Mile               0.0116      0.013      0.858      0.391      -0.015       0.038
```

```
# this is the statsmodels approach, need to add constant
Xte1 = sm.add_constant(Xte[:, :14])
y_hat_te = model.predict(Xte1)

test_RMSE_14movies = np.sqrt(np.mean((Yte - y_hat_te)**2))

print(f'Test RMSE with the mean: {test_RMSE_null: .2f}')
print(f'Test RMSE with Sweet Home Alabama: {test_RMSE_1movie: .2f}')
print(f'Test RMSE with 14 movies: {test_RMSE_14movies: .2f}')
```

```
Test RMSE with the mean:  0.97
Test RMSE with Sweet Home Alabama:  0.88
Test RMSE with 14 movies:  0.81
```

## What is the model now?

# Even more adventurous: use all 99 movies

What to do about missing? Let's keep as 0 for now (did not rate = hate…)

**Dealing with missing values is an important topic, that we won't cover here**

```
Xtr[np.isnan(Xtr)] = 0
Xtr1 = sm.add_constant(Xtr)
model = sm.OLS(Ytr, Xtr1)
model = model.fit()

Xte[np.isnan(Xte)]=0
Xte1 = sm.add_constant(Xte)
y_hat_te = model.predict(Xte1)

test_RMSE_99movies = np.sqrt(np.mean((Yte - y_hat_te)**2))

print(f'Test RMSE with the mean: {test_RMSE_null: .2f}')
print(f'Test RMSE with Sweet Home Alabama: {test_RMSE_1movie: .2f}')
print(f'Test RMSE with 14 movies: {test_RMSE_14movies: .2f}')
print(f'Test RMSE with 99 movies: {test_RMSE_99movies: .2f}')
```

```
Test RMSE with the mean:  0.97
Test RMSE with Sweet Home Alabama:  0.88
Test RMSE with 14 movies:  0.81
Test RMSE with 99 movies:  0.78
```

# Linear Regression - Fitting the Model

INTRODUCTION

TO DATA SCIENCE

# Least squares regression: fitting the model

- Let's start from the simple case $p = 1$: one feature (Sweet Home Alabama) + constant/intercept
- Finding the coefficients:

$$\min_{\beta_0, \beta_1} \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_i))^2$$

- Solution (we won't prove here):

$$\hat{\beta}_1 = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2} = \frac{\widehat{Cov}(X, Y)}{\widehat{Var}(X)}, \quad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

Does $\hat{\beta}_1$ look familiar?

# General algebric solution for any $p$

- Write our problem in matrix-vector notation (now $\beta \in \mathbb{R}^{p+1}$ is vector of coefficients):

$$\min_{\beta} RSS(\beta) = \min_{\beta} \|Y - X\beta\|^2$$

- This is a quardratic function of $\beta$, find minimizer by differentiating and equating to zero. Normal equations:

$$-2X^T(Y - X\beta) = 0$$

- This looks scary, but it simply means:

$$\frac{\partial RSS(\beta)}{\partial \beta_j} = \sum_{i=1}^{n} x_{ij} \left( y_i - (\beta_0 + \sum_{k=1}^{p} x_{ik}\beta_k) \right) = 0, \quad j = 0, \ldots, p$$

- The problem:

$$-2X^T(Y - X\beta) = 0$$

- The solution:

$$X^T X\beta = X^T Y \quad \Rightarrow \quad \hat{\beta} = (X^T X)^{-1} X^T Y.$$

(the second derivative matrix is positive definite $\Rightarrow$ minimum)

- For $p = 1$ we would recover back exactly the formulas from before

# A geometric view

- The columns of the matrix $X_{n\times(p+1)}$ are vectors $X_0^c, \ldots, X_p^c \in \mathbb{R}^n$.
  Each feature in $Tr$ is such a vector.
- The response vector in $Tr$ is $Y_{n\times 1}$, which is also a vector in $\mathbb{R}^n$.

- $X\beta = X_0^c\beta_0 + \cdots + X_p^c\beta_p$ is a linear combination of the columns.
- Hence, in $\min_\beta \|Y - X\beta\|^2$ we are seeking a linear combination of the columns which is closest to $Y$ in $\mathrm{Span}(X_0^c, \ldots, X_p^c)$.

$\Rightarrow$ OLS is an *orthogonal projection* of $Y$ on the column space of $X$

# Linear Regression - Statistical Perspective

INTRODUCTION

TO DATA SCIENCE

# A statistical model for inference

- So far we did not assume any specific *true* relationship between $y$ and $x$
- Let us now *assume* the following model:

$$y = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p + \epsilon, \quad \epsilon \sim N(0, \sigma^2)$$

1. $E(y|x) = x^T \beta$ is a linear function of $x$
2. The error $(y - E(y|x))$ has a normal distribution and is independent for each observation

- If this assumption holds, we can investigate the distribution of $\hat{\beta}$ and use that to do inference on the model

# Distribution of the OLS solution under the model assumptions

- What we know:

$$(a)\ E(Y) = X\beta, \quad (b)\ Cov(Y) = \sigma^2 I_n, \quad (c)\ \hat{\beta} = (X^T X)^{-1} X^T Y$$

- Mean:

$$E(\hat{\beta}) \overset{(c)}{=} (X^T X)^{-1} X^T E(Y) \overset{(a)}{=} (X^T X)^{-1} X^T X\beta = \beta.$$

- Covariance matrix:

$$Cov(\hat{\beta}) \overset{(c)}{=} (X^T X)^{-1} X^T Cov(Y) X (X^T X)^{-1} \overset{(b)}{=} \sigma^2 (X^T X)^{-1} (X^T X)(X^T X)^{-1} = \sigma^2$$

# Statistical inference

- From the previous formulas we conclude: $\hat{\beta}_j \sim N(\beta_j, \sigma^2 (X^T X)^{-1}_{j,j})$.
- Recall that our second goal (beyond prediction) was *inference*: which variables are important?
- Now we can formalize this as a hypothesis test: for each variable $j$, test the null $H_{0j} : \beta_j = 0$.
- If $H_{0j}$ holds, then $\hat{\beta}_j \sim N(0, \sigma^2 (X^T X)^{-1}_{j,j})$.
- Assuming $\sigma^2$ is known, this leads to a simple $Z$-test as we studied
- Since $\sigma^2$ is not known, we need to estimate it and get a T-test instead (details omitted).

# Back to the 14-movies model, now with the inference:

```
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.276
Model:                            OLS   Adj. R-squared:                  0.274
Method:                 Least Squares   F-statistic:                     217.0
Date:                Mon, 14 Aug 2023   Prob (F-statistic):               0.00
Time:                        11:02:03   Log-Likelihood:                -9694.0
No. Observations:                8000   AIC:                         1.942e+04
Df Residuals:                    7985   BIC:                         1.952e+04
Df Model:                          14
Covariance Type:            nonrobust
================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const                        0.4074      0.090      4.509      0.000       0.230       0.584
Independence Day             0.0668      0.013      4.996      0.000       0.041       0.093
The Patriot                 -0.0161      0.012     -1.382      0.167      -0.039       0.007
The Day After Tomorrow       0.0503      0.011      4.612      0.000       0.029       0.072
Pirates of the Caribbean     0.0642      0.012      5.375      0.000       0.041       0.088
Pretty Woman                 0.1522      0.012     13.135      0.000       0.129       0.175
Forrest Gump                -0.0673      0.014     -4.953      0.000      -0.094      -0.041
The Green Mile               0.0116      0.013      0.858      0.391      -0.015       0.038
```

# OLS regression summary

- Minimize RSS on $Tr$ to find the "best" linear fit for $Y$ as a function of $X$
- Algebraic solution, geometric interpretation: projection
- Under the assumed statistical model (strong assumptions!) can do inference on which variables are important
- The most important tool in the statistical/predictive modeling toolbox!
- Learn more: Statistical Models course in Statistics

# Comment I: OLS Interpretation

- As we just saw, under the statistical model,
  $E\hat{\beta} = \beta \;\Rightarrow\; E(\hat{y}|x) = x^T E(\hat{\beta}) = x^T \beta = E(y|x).$

- Even when the model doesn't hold, the use of RSS / squared error loss implies estimation of conditional expectation (details omitted)
- Hence an interpretation of the OLS prediction is an *attempt* to estimate the conditional expectation $E(y|x)$

- This conditional expectation is clearly interesting: it summarizes what we learned about $y$ from seeing $x$
- The attempt may not be successful, if the model is not so good (more on that later), but at least we know what we are trying to predict!

# Comment II: OLS via Likelihood

- Recall the assumed model:

$$y = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p + \epsilon, \quad \epsilon \sim N(0, \sigma^2)$$

- An alternative criterion to *maximize* in order to get $\hat{\beta}$: the Likelihood of the data

$$L(\beta|X, y) = \prod_{i=1}^{n} f(y_i|X; \beta)$$

- The $f$ being the Normal distribution density
- This can be shown to give the exact same solution to $\hat{\beta}$!

# Logistic Regression

INTRODUCTION

TO DATA SCIENCE

# What about classification?

- We will focus on the simplest (and most important) case of two-class classification:
  - Impressionist vs. realist
  - Sick vs healthy
  - Buy vs don't buy
- As before, we have $Tr = (X, Y)$ of size $n$, $Te$ of size $m$.
- For now, keep assuming $x \in \mathbb{R}^p$ is numeric as in the wikiart paintings example
- Can we use the OLS mechanism we have built to build a classification model?
- For sure we can, if we encode $y = \text{impressionist} \Rightarrow y = 1, \quad y = \text{realist} \Rightarrow y = 0$, we have numeric $y$

# What is wrong with using OLS for classification?

- If we encode $y$ as above what is $E(y|x)$? It is $P(y = \text{impressionist}|\ \text{image})$ — a clearly interesting quantity
- Problem: as a probability, $0 \leq P(y = \text{impressionist}|\ \text{image}) \leq 1$. But model predictions $x^T \hat{\beta}$ can fall outside the legal range!
- Another problem: can we make the model assumptions of normal $\epsilon$? No — because $y$ can only be $0$ or $1$
- The idea: try to create an approach that is similar to OLS, but more fitting for classification, taking into account the limited range of values and the need for a sensible statistical model

# Logistic regression

- Deals with the two problems above
- We start from assuming a model:

$$\log \frac{P(y=1|x)}{P(y=0|x)} = \log \frac{P(y=1|x)}{1-P(y=1|x)} = \text{logit}(P(y=1|x)) = x^T\beta$$

- Notice that now all values are legal:

$$0 \leq P(y=1|x) \leq 1 \quad \Leftrightarrow \quad -\infty \leq \log \frac{P(y=1|x)}{P(y=0|x)} \leq \infty.$$

- Another way of writing this:

$$P(y=1|x) = \frac{\exp(x^T\beta)}{1+\exp(x^T\beta)} \qquad P(y=0|x) = 1 - P(y=1|x) = \frac{1}{1+\exp(x^T\beta)}$$

# Fitting a logistic regression

- Given training data $Tr$, we want to find the best coefficients $\hat{\beta}$
- This is done by maximum likelihood, finding $\beta$ to maximize:

$$L(\beta|X,y) = \prod_{i=1}^{n} P(y_i|x_i;\beta) = \prod_{i=1}^{n} P(y_i = 1|x_i;\beta)^{y_i} P(y_i = 0|x_i;\beta)^{1-y_i}$$

$$\max_{\beta} \prod_{i=1}^{n} \left( \frac{\exp(x_i^T \beta)}{1 + \exp(x_i^T \beta)} \right)^{y_i} \left( \frac{1}{1 + \exp(x_i^T \beta)} \right)^{1-y_i}$$

- The solution is $\hat{\beta}$, the logistic regression coefficients estimates
- Predicting on $x \in Te$:

$$\widehat{P(y=1|x)} = \frac{\exp(x^T\hat{\beta})}{1+\exp(x^T\hat{\beta})} \quad \Rightarrow \quad \hat{y} = \begin{cases} 1 & \text{if } \widehat{P(y=1|x)} > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

# Interpretation of coefficients

- We can write our model as:

$$\log \frac{P(y=1|x)}{P(y=0|x)} = x^T \beta$$

- The expression on the left is called the *log odds*: log of the ratio of positive vs negative probability
- Interpretation: $\beta_j$ is the change in the log odds from a change of 1 unit in $x_j$.
- For example, if $\beta_j = 1$ then when $x_j = 1$ vs $x_j = 0$ the log odds increase by $1$, so the odds increase times $e = 2.72$, which is roughly the increase in $P(y=1|x)$ when it is close to $0$.

# Example: SAHeart

INTRODUCTION

TO DATA SCIENCE

# Example: South African Hearth Disease Data

```
saheart = pd.read_table("../datasets/SAheart.data", header = 0, sep=',', index_col=0)

print(saheart.describe())
```

```
              sbp     tobacco         ldl   adiposity       typea     obesity  \
count  462.000000  462.000000  462.000000  462.000000  462.000000  462.000000
mean   138.326840    3.635649    4.740325   25.406732   53.103896   26.044113
std     20.496317    4.593024    2.070909    7.780699    9.817534    4.213680
min    101.000000    0.000000    0.980000    6.740000   13.000000   14.700000
25%    124.000000    0.052500    3.282500   19.775000   47.000000   22.985000
50%    134.000000    2.000000    4.340000   26.115000   53.000000   25.805000
75%    148.000000    5.500000    5.790000   31.227500   60.000000   28.497500
max    218.000000   31.200000   15.330000   42.490000   78.000000   46.580000

          alcohol         age         chd
count  462.000000  462.000000  462.000000
mean    17.044394   42.816017    0.346320
std     24.481059   14.608956    0.476313
min      0.000000   15.000000    0.000000
25%      0.510000   31.000000    0.000000
50%      7.510000   45.000000    0.000000
75%     23.892500   55.000000    1.000000
max    147.190000   64.000000    1.000000
```

# SAHeart: Data Splitting with SKlearn

```
saheart_X=pd.get_dummies(saheart.iloc[:, :9]).iloc[:, :9]
saheart_y=saheart.iloc[:, 9]

from sklearn.model_selection import train_test_split

Xtr, Xte, Ytr, Yte = train_test_split(saheart_X, saheart_y, test_size=0.2, random_s

print(f'No. of train rows: {Xtr.shape[0]}, no. train of cols: {Xtr.shape[1]}')
print(f'No. of test rows: {Xte.shape[0]}, no. test of cols: {Xte.shape[1]}')
print(f'no. of obs in train y: {Ytr.shape[0]}')
print(f'no. of obs in test y: {Yte.shape[0]}')
```

```
No. of train rows: 369, no. train of cols: 9
No. of test rows: 93, no. test of cols: 9
no. of obs in train y: 369
no. of obs in test y: 93
```

# SAHeart: LR with statsomdels

```
import statsmodels.api as sm

model = sm.Logit(Ytr, sm.add_constant(Xtr))
model = model.fit()

print(model.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.513971
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:                    chd   No. Observations:                  369
Model:                          Logit   Df Residuals:                      359
Method:                           MLE   Df Model:                            9
Date:                Mon, 14 Aug 2023   Pseudo R-squ.:                  0.1994
Time:                        11:02:03   Log-Likelihood:                -189.66
converged:                       True   LL-Null:                       -236.90
Covariance Type:            nonrobust   LLR p-value:                 2.041e-16
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -5.3373      1.478     -3.612      0.000      -8.234      -2.441
sbp            0.0081      0.006      1.262      0.207      -0.004       0.021
tobacco        0.0563      0.029      1.926      0.054      -0.001       0.114
ldl            0.1676      0.064      2.599      0.009       0.041       0.294
adiposity      0.0273      0.033      0.823      0.411      -0.038       0.092
typea          0.0411      0.014      3.005      0.003       0.014       0.068
obesity       -0.0820      0.049     -1.661      0.097      -0.179       0.015
```

# SAHeart: LR with SKlearn

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(solver='lbfgs', max_iter=10000)
model.fit(Xtr, Ytr)

print('intercept:', model.intercept_)
print('coef:', model.coef_)
```

```
intercept: [-5.37683503]
coef: [[ 0.0080158   0.0557117   0.16737956  0.0271416   0.0410251  -0.08127558
   0.00199583  0.04753859 -0.76405442]]
```

# SAHeart: LR Test Performance

```
 1 from sklearn.metrics import confusion_matrix
 2
 3 p_hat_te = model.predict_proba(Xte)[:, 1]
 4 y_hat_te = p_hat_te > 0.5
 5 conf = confusion_matrix(Yte, y_hat_te)
 6
 7 pd.DataFrame(
 8   confusion_matrix(Yte, y_hat_te),
 9   index=['true:no', 'true:yes'],
10   columns=['pred:no', 'pred:yes']
11 )
```

| | pred:no | pred:yes |
|---|---|---|
| true:no | 52 | 7 |
| true:yes | 15 | 19 |

```
acc = np.mean(Yte == y_hat_te)
err = np.mean(Yte != y_hat_te)
print(f'Accuracy: {acc: .2f}, Misclassification loss: {err: .2f}')
```

```
Accuracy:  0.76, Misclassification loss:  0.24
```

# Classification Model Evaluation

INTRODUCTION

To Data Science

# Measuring Classification Performance

- Different errors have different costs/value.
- Summarize performance in different ways that capture different types of errors:

| | **Pred** | | |
|---|---|---|---|
| **Real** | Pos | Neg | Total |
| Pos | $TP$ | $FN$ | $P$ |
| Neg | $FP$ | $TN$ | $N$ |
| Total | $\hat{P}$ | $\hat{N}$ | |

$P = \sum_{i=n+1}^{n+m} y_i$ number of positive examples, similarly $N$.

$\hat{P} = \sum_{i=n+1}^{n+m} \hat{y}_i$ number of positive predictions, similarly $\hat{N}$.

$TP = \sum_{i=n+1}^{n+m} y_i \hat{y}_i$ number of true positives, $FP = \hat{P} - TP$

$TN = \sum_{i=n+1}^{n+m} (1 - y_i)(1 - \hat{y}_i)$ number of true negatives, $FN = \hat{N} - TN$

| | **Pred** | | |
|---|---|---|---|
| **Real** | Pos | Neg | Total |
| Pos | $TP$ | $FN$ | $P$ |
| Neg | $FP$ | $TN$ | $N$ |
| Total | $\hat{P}$ | $\hat{N}$ | $m$ |

---

Accuracy: $P(Correct) = (TN + TP)/m$

Prediction error: $P(Error) = (FN + FP)/m$

Precision+ (positive predictive value): $P(True+|Pred+) = TP/\hat{P}$

Recall+ (sensitivity, true positive rate): $P(Pred+|True+) = TP/P$

False positive rate: $P(Pred+|True-) = FP/N$

Harmonic mean of precision and recall: $F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$

```
pd.DataFrame(
  confusion_matrix(Yte, y_hat_te),
  index=['true:no', 'true:yes'],
  columns=['pred:no', 'pred:yes']
)
```

| | pred:no | pred:yes |
|---|---|---|
| true:no | 52 | 7 |
| true:yes | 15 | 19 |

```
from sklearn.metrics import classification_report

print(classification_report(Yte, y_hat_te))
```

```
              precision    recall  f1-score   support

           0       0.78      0.88      0.83        59
           1       0.73      0.56      0.63        34

    accuracy                           0.76        93
   macro avg       0.75      0.72      0.73        93
weighted avg       0.76      0.76      0.76        93
```

All is still based on that cutoff, 0.5!

# Classification evaluation: different goals

We can think of several different prediction goals, all potentially important:

1. Classify correctly — make few (weighted) errors on test set or new prediction points
2. Predict probabilities well: $\widehat{P(y=1|x)} \approx P(y=1|x)$ for new points
3. Rank well: given multiple prediction points, predict which one is *more likely* to have $y=1$.

These different tasks can reflect in the loss function / model evaluation task:

1. Correct classification: misclassification loss as above, also precision, recall etc.
2. Good probability prediction: using Bernoulli loss / cross entropy:

$$L(y, \hat{p}) = \hat{p}^y (1-\hat{p})^{(1-y)}$$

3. How do we measure ranking perofrmance of a model on a test set?

# The ROC Curve

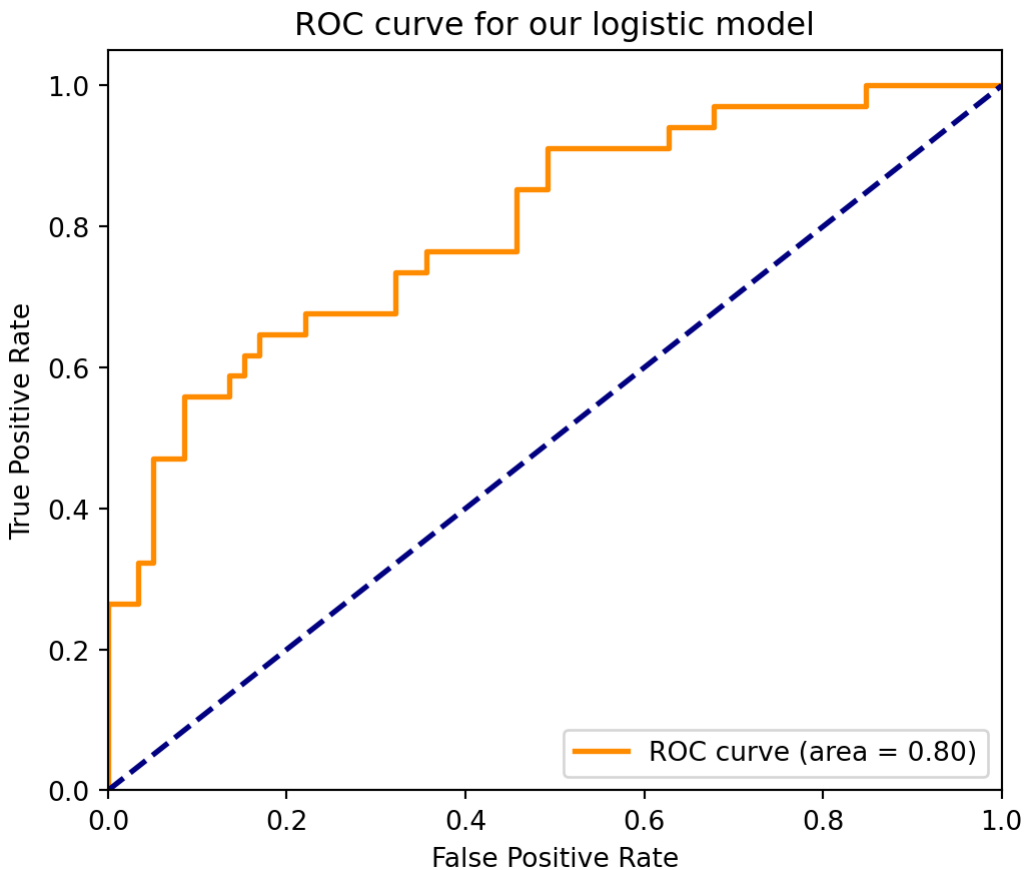
The idea: to evaluate ranking performance, do not set the threshold $0.5$ but check what happens at all possible thresholds:

1. True positive rate: what % of the positive observations pass the threshold?
2. False positive rate: what % of the negative observations pass the threshold?

- The ROC curve plots TPR vs FPR for all possible threholds: if the model ranks well, for high thresholds we will have $FPR \approx 0$, while for low thresholds we will have $TPR \approx 1$

- Note that even if $\widehat{P(y = 1|x)}$ predicts probabilities badly, or even if the predictions are not in the range $[0, 1]$, the ranking can still be good

## ▶ Code


ROC curve for our logistic model
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.80)

# The Area Under the Curve (AUC)

- *For a random ranking:* $FPR \approx TPR$ at every threshold, so we are around the diagonal $x = y$:

$$AUC \approx 0.5$$

- *For a perfect ranking model:* at high thresholds, $FPR = 0$, at low thresholds $TPR = 1$, hence:

$$AUC = 1.$$

- Very nice interpretation of AUC: Assume the test set has $m_1$ ones ($y = 1$) and $m_0$ zeros, then AUC is the % of correctly ranked pairs with different response:

$$AUC = \frac{\#\left\{(i,j) : y_i = 0, y_j = 1 \text{ and } \hat{p}_i < \hat{p}_j\right\}}{m_1 \times m_0}$$

# But think.

- Where did our paintings go?
- Is logistic regression a suitable model for our paintings images?

Intro to Data Science