databites.tech @rfeers Josep Ferrer

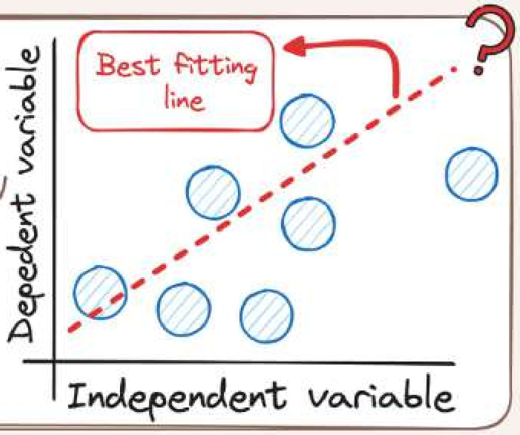
ML BASICS - SIMPLE LINEAR REGRESSION THEORY

#1 SIMPLE LINEAR REGRESSION

Linear regression is the simplest statistical regression method used for predictive analysis.
- The most common is the SIMPLE LINEAR REGRESSION
  1 independent variable + 1 dependent variable

The main goal?

Find a linear relationship between the independent variable (predictor) and the dependent (output)

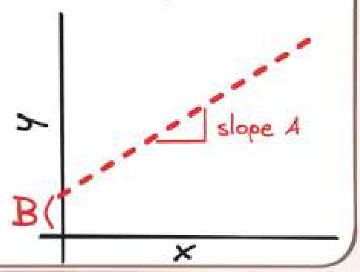


#2 HOW TO COMPUTE IT?

To compute the best-fit line linear regression, we use the line function.

$$Y_i = Ax + B$$

Y_i = Dependent Variable
B = intercept
A = slope
x_i = Independent variable



#3 HOW TO DEFINE THE BEST FIT?

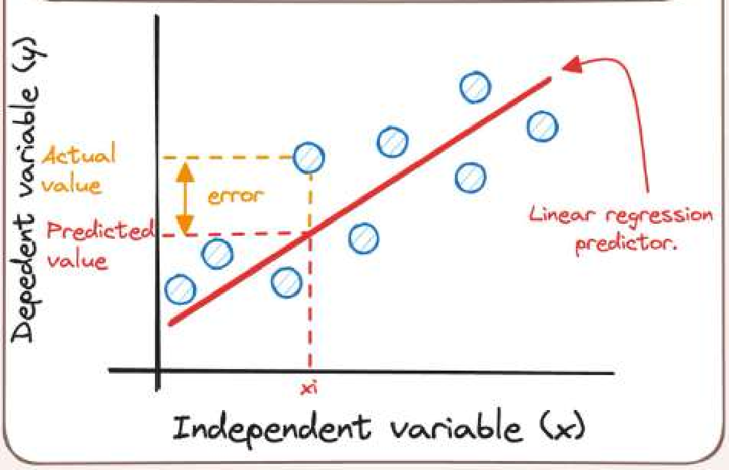
We define the best fit line as the line that presents the least error.

The error between predicted values and the actual values should be minimum.

Random Errors (Residuals)

Residuals are defined as the difference between observed values of the dependent variable and the predicted ones.

$$\epsilon_i = y_{predicted} - y_i$$



#4 HOW TO OBTAIN IT MATHEMATICALLY?

We use a cost function that helps us work out the optimal values for A and B.

MEAN SQUARED ERROR (MSE)

We use the average of the squared error that occurs between predicted and observed values.

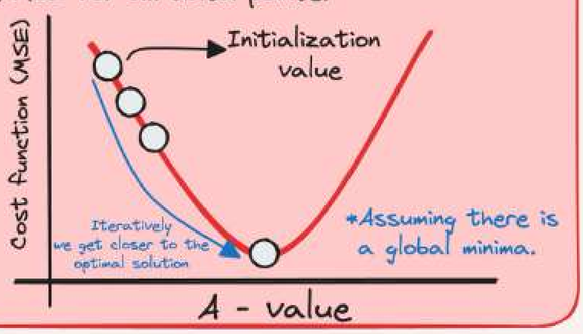
$$MSE = \frac{1}{N}\sum_{i=1}^{n}(y_i - (Ax + B))^2$$

To find the optimal solution

GRADIENT DESCENT

Gradient descent is one of the optimization algorithms that optimizes the cost function.

To obtain the optimal solution, we need to reduce MSE for all data points.



#5 EVALUATION

The most used metrics are,
- Coefficient of Determination or R-Squared (R2)
- Root Mean Squared Error (RMSE)

#6 ASSUMPTIONS TO APPLY IT

1. Linearity of the variables:
   There needs to be linear dependency between the dependent and the independent variables.

2. Independence of residuals:
   The error terms should not be dependent on one another

3. Normal distribution of residuals
   The mean of residuals should follow a normal distribution with a mean close to zero.

4. The equal variance of residuals.
   The error terms must have constant variance.

databites.tech @rfeers Josep Ferrer

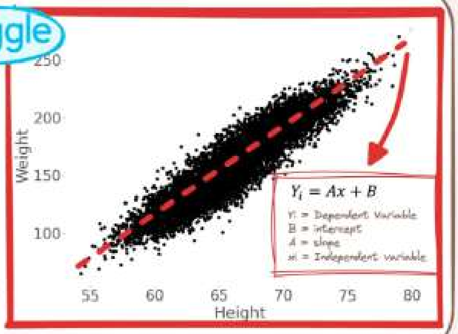
ML BASICS - SIMPLE LINEAR REGRESSION EXEMPLIFIED

#1 GETTING THE DATA

kaggle

Today we are dealing with some real-world data. And the turn is for... height and weight!

One of the classic examples of linear dependency.



#2 SOME FIRST ANALYSIS

We first check the table. It has 3 main variables:

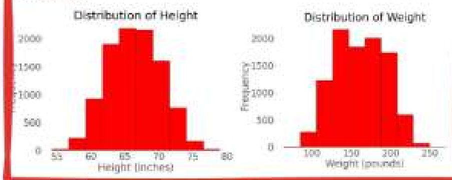
- Gender
- Height
- Weight

	Gender	Height	Weight
0	Male	73.847017	241.893563
1	Male	68.781904	162.310473
2	Male	74.110105	212.740856
3	Male	71.730978	220.042470
4	Male	69.881796	206.349801

Check the size of the table and the number of nulls with df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Gender  10000 non-null  object
 1   Height  10000 non-null  float64
 2   Weight  10000 non-null  float64
dtypes: float64(2), object(1)
memory usage: 234.5+ KB
```

Finally check the distribution of our data!



Both present a normal distribution.

APPROACH 1 - GRADIENT DESCENDENT

We use the mean square error (MSE) that occurs between predicted and observed values.

$$MSE = \frac{1}{N}\sum_{i=1}^{n}(y_i - (Ax + B))^2$$

This translates into defining two main functions:

```
def compute_mse(y_true, y_pred):
    N = len(y_pred)
    MSE = np.mean((y_true - y_pred) ** 2)/N
    return MSE
```

The function to compute MSE

```
def gradient_descent(x, y, A, B, learning_rate):
    y_pred = A * x + B
    dA = -2 * np.sum(x * (y - y_pred)) / N
    dB = -2 * np.sum(y - y_pred) / N
    A -= learning_rate * dm
    B -= learning_rate * db
    return A, B
```

The function to update A and B

We initialize our code with:

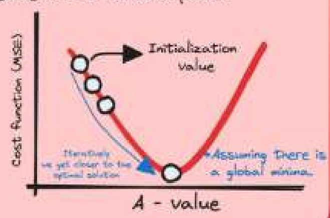
- A = 0
- B = 0
- A learning rate of 0.0001
- A max number of iterations

The learning rate allows the algorithm to learn faster or slower.

GRADIENT DESCENT

Gradient descent is one of the optimization algorithms that optimizes the cost function.

To obtain the optimal solution, we need to reduce MSE for all data points.



APPROACH 2 - OLS (Ordinary Least Squares)

The goal of OLS is to find the values of A and B that minimize the sum of the squared residuals (S).

$$S = \sum_{i=1}^{N}(y_i - (Ax_i + B))^2$$

To minimize S, we can easily take its partial derivatives and set them to zero.

$$\frac{\delta S}{\delta A} = 0 \quad \frac{\delta S}{\delta B} = 0$$

Solving these two equations we obtain a closed mathematical soltuion.

$$A = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

$$B = \bar{y} - B\bar{x}$$

This translates into defining some lines of code to find this mathematical closed solution:

```
x_mean = np.mean(x)
y_mean = np.mean(y)

for i in range(N):
    numerator += (x[i] - x_mean) * (y[i] - y_mean)
    denominator += (x[i] - x_mean) ** 2

A = numerator / denominator
B = y_mean - (A * x_mean)
```

APPROACH 3 - SCI-KIT LEARN

Scikit-learn is a versatile Python library offering a wide range of machine learning tools, including algorithms for:

- Classification
- Regression
- Clustering
- And way more...

```
from sklearn.linear_model import LinearRegression

# create linear regression object
lr = LinearRegression()

# fit linear regression
lr.fit(df[['Height']], df['Weight'])

x = np.linspace(55,80,100)
y_predicted = lr.coef_*x + lr.intercept_
```

Import the library

Create a Linear Regression object

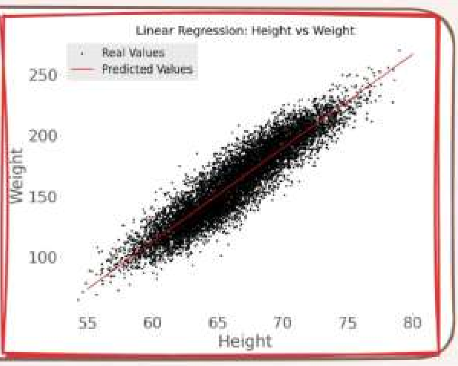
Train it with our data

Get the predicted output.

#4 FINAL RESULTS

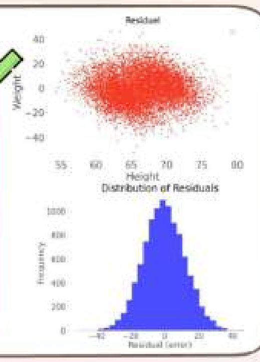
A = 7.71728764
B =-350.737191812137

$$Y_i = Ax + B$$



ASSUMPTIONS

1. Linearity of the variables
2. Independence of residuals
3. Normal distribution of residuals
4. The equal variance of residuals



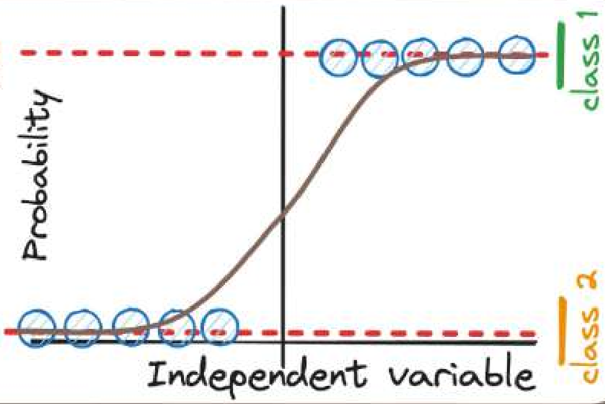
ML BASICS - LOGISTIC REGRESSIONS

#1 LOGISTIC REGRESSION

Logistic regression is used for binary classification problems (two categories). Can be extended for more classes too!

The main goal?

Logistic regression aims to find the probability that a given input belongs to a certain class.
The predicted output is categorical!



#2 HOW TO COMPUTE IT?

A sigmoid function maps any real-valued number into a value between 0 and 1, suitable for probability interpretation.

$$\sigma(x) = \frac{1}{1 + e^{-Ax-B}}$$

#3 HOW TO DEFINE THE BEST FIT?

For every parametric machine learning algorithm, we need a **LOSS FUNCTION**

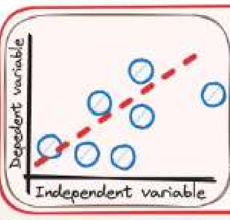
We want to minimize it

To find the global minimum.

Determine the optimal parameters → $A*$ $B*$

BONUS - FROM LINEAR TO LOGISTIC REGRESSION

To obtain the sigmoid function, we start from the linear regression equation.



1. We start with the Linear Regression equation. $Y = Ax+B$

We want a probability.

2. This probability might be bigger than 1! $P = Ax + B$

We take odds. $\frac{P}{1-P} = Ax + B$

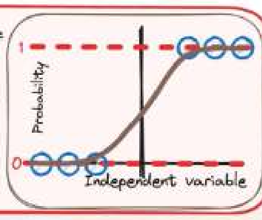
$\log\left(\frac{P}{1-P}\right) = Ax + B$

$\frac{P}{1-P} = e^{(Ax+B)}$

$P = \frac{1}{1 - e^{-Ax+B}}$

The sigmoid function!

We get the logistic regression function!



#4 HOW TO OBTAIN IT MATHEMATICALLY?

For a binary classification problem, the model output corresponds the probability of prediction y being:

- 1 -> For a class.
- 0 -> For the other class.

or vice versa!

If we define as our hypothesis: $P(y = 1|x; A; B) = y_{hat}$

We know as well that... $P(y = 0|x; A; B) = 1 - y_{hat}$

Considering this both equations, the loss function to minimize can be obtained.

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m}[(y^{(i)}\log(\hat{y}^{(i)}) + (1 - y^{(i)})\log(1 - \hat{y}^{(i)})]$$

(Binary Coss-Entropy Loss or the Log Loss function)

By looking at the Loss function we see:
- The loss approaches 0 when we predict correctly.
- The loss function approaches infinity if we predict incorrectly.

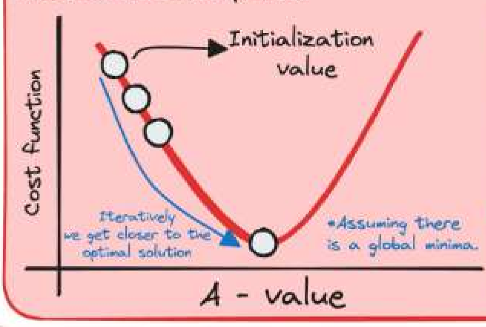
#5 FIND THE OPTIMAL SOLUTION

Now that we know our hypothesis function and the loss function.

GRADIENT DESCENT

It is one of the most used algorithms to optimize our cost function.

To obtain the optimal solution, we reduce the cost function all data points.



$A = A - lr * dA$
$B = B - lr * dB$

lr is the learning rate

Partial derivatives of the cost function

#6 MODEL EVALUATION

- **Confusion Matrix:**
  A table used to describe the performance of a classification model.
- **ROC Curve:**
  A graph showing the performance of a classification model at all classification thresholds.
- **AUC:**
  The area under the ROC curve; a higher AUC indicates a better model.

#7 ASSUMPTIONS OF THE MODEL

1. **Binary Outcome:**
   The dependent variable is binary.
2. **Linearity in Log Odds:**
   The log odds of the outcome is modeled as a linear relationship with the independent variables.
3. **No Multicollinearity:**
   Independent variables should not be highly correlated with each other.
4. **Large Sample Size:**
   Requires a sufficiently large sample size for reliable results.

ML BASICS - SIMPLE LOGISTIC REGRESSION EXEMPLIFIED

#1 GETTING THE DATA

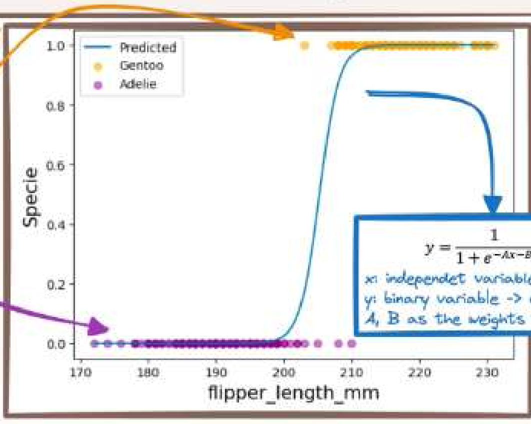
kaggle

Today we are dealing with some real-world data. And the turn is for... penguin species classification!

We have two cuties that need to be recognized.

Our friend Gentoo! It equals y = 1

Our friend Adelie! It equals y = 0



$$y = \frac{1}{1 + e^{-Ax-B}}$$

x: independet variable
y: binary variable -> class
A, B as the weights to find.

#2 SOME FIRST ANALYSIS

We first check the table.
It has 6 main variables:

- species
- island
- bill_length
- bill_depth_mm
- flipper_length_mm
- body_mass_g

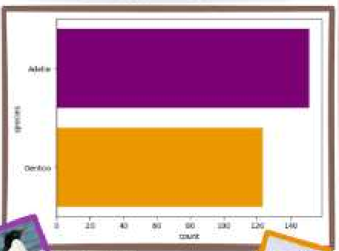
Then we check the size of the table and the number of nulls with

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 274 entries, 0 to 273
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   species            274 non-null    object
 1   island             274 non-null    object
 2   bill_length_mm     274 non-null    float64
 3   bill_depth_mm      274 non-null    float64
 4   flipper_length_mm  274 non-null    float64
 5   body_mass_g        274 non-null    float64
dtypes: float64(4), object(2)
memory usage: 13.0+ KB
```

Finaly we check how many data we have for each specie.

```
sns.countplot(df["species"],palette='Set2')
```



151 Adelie

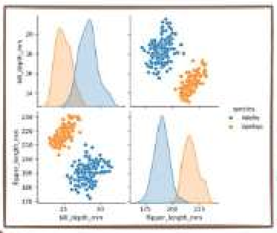
123 Gentoo

#3 UNDERSTANDING OUR DATA

```
import seaborn as sns
sns.heatmap(numeric_df.corr(), annot=True, fmt=".0%")
```

This script is used to visually represent the correlations between all pairs of numeric columns in a DataFrame.

The correlation range is 1 to -1 and indicates the strength and direction of their relationship.



```
sns.pairplot(df_temp.iloc[:,:],hue='species')
```

It generates a grid of scatter plots, each showing the relationship between two different variables in the dataset and with colors based on species.

This provides an overview of how different variables interact and how these interactions vary by species.

#4 PREPARATION OF THE DATASET

First, we need to define our specie variable as a binary one. (dummy variable)

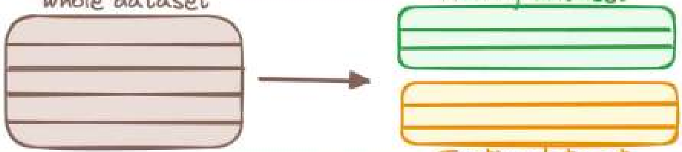
```
pd.get_dummies(df, dtype=int)
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	species_Adelie	species_Gentoo
27	37.9	18.6	172.0	3150.0	1	0
19	37.8	18.3	174.0	3400.0	1	0
121	40.2	17.0	176.0	3450.0	1	0

Then we need to split our dataset into training and testing.

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.5, random_state=41)
```

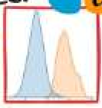


#5 Classification I - 1 variable to 2 classes.

```
X = df_dummy[['flipper_length_mm']]
Y = df_dummy["species_Gentoo"]
```

We select a single variable (flippery_length)

The variable with the most distinct distribution across both species.



```
from sklearn.linear_model import LogisticRegression
# Fit the Logistic Regression model
log = LogisticRegression()

log.fit(X_train, Y_train)

def sigmoid(x, A, B):
    return 1 / (1 + np.exp(-A*x - B))

Y_predicted = sigmoid(X_test, A, B)

# Checking our accuracy
accuracy = log.score(X_test, Y_test)
```

- Import the library
- Create a Logistic Regression object
- Train the model
- Define the sigmoid function
- Predict with the testing data
- Check the accuracy

#6 Classification II - 2 variables to 2 classes.

```
X = df_dummy.iloc[:, 1:3]
Y = df_dummy["species_Gentoo"]
```

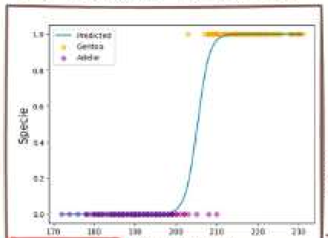
We select a two variables
- flippery_length
- bill_depth

The variable:
- Less correlated (-55%)
- Easiest to differentiate.

```
from sklearn.linear_model import LogisticRegression
# Fit the Logistic Regression model
log = LogisticRegression()

log.fit(X_train, Y_train)

Y_predicted = log.predict(X_test)

# Checking our accuracy
accuracy = log.score(X_test, Y_test)
```

- Import the library
- Create a Logistic Regression object
- Train the model
- Predict with the testing data
- Check the accuracy

#7 FINAL RESULT



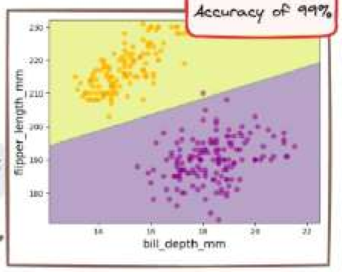
Accuracy of 98%

$$y = \frac{1}{1 + e^{-Ax-B}}$$

A: 0.65
B: -133.78

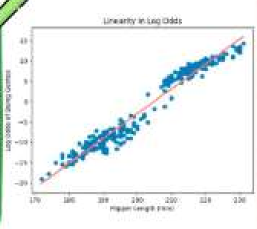
A: -1.2, 0.5
B: -83.2

Accuracy of 99%



#8 ASSUMPTIONS

1. Binary Outcome
2. Linearity in Log Odds
3. No Multicollinearity
4. Sample Size



ML BASICS - LOGISTIC REGRESSION WITH MULTIPLE CLASSES

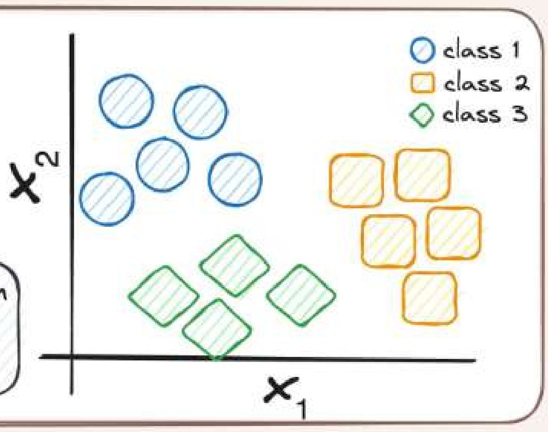
LOGISTIC REGRESSION WITH MULTIPLE CLASSES

By default Logistic Regression is limited to two-class classification problems.

To adapt it into a multiple class classification model, we have two options:

One-vs-Rest (OvR): For each class, fit a logistic regression model to distinguish that class from all other classes.

Multinomial logistic regression: Extend the Logistic Regression model from a simple binary classification to multiple-class classification.

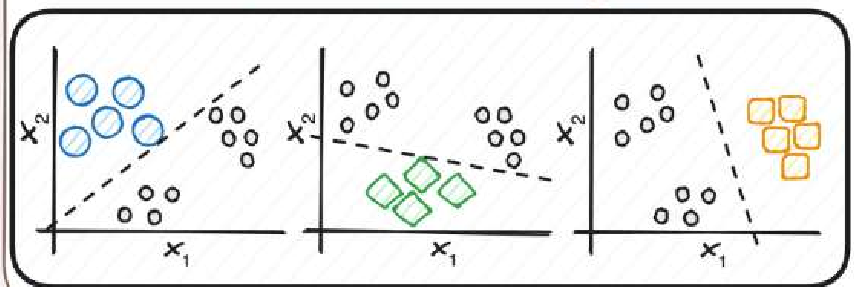


#1 ONE VS REST (OvR)

As many classifiers as many classes we have.

Strategy: Train multiple Logistic Regression classifiers.

Each classifier focuses on a single class (and considers the other classes as a single one)



We get three classification models:

$f_1(x\,;\,w_1)$ ○ ⟷ 0

$f_2(x\,;\,w_2)$ ◇ ⟷ 0

$f_3(x\,;\,w_3)$ □ ⟷ 0

And the final output is given by

The class with the highest score.

$$\underset{n}{\operatorname{argmax}}\; f_n(x\,;\,w_n)$$

#2 MULTINOMIAL LOGISTIC REGRESSION

Instead of assuming that we only have two classes (0 or 1)

We build a model that outputs a vector of probabilities for each class.

$$\cancel{\begin{aligned} P(y=1|x;A;B) &= y_{hat} \\ P(y=0|x;A;B) &= 1-y_{hat} \end{aligned}}$$

$$f(\mathbf{x};\mathbf{W}) = \begin{bmatrix} P(y=1|\mathbf{x};\mathbf{w}_1) \\ P(y=2|\mathbf{x};\mathbf{w}_2) \\ \vdots \\ P(y=K|\mathbf{x};\mathbf{w}_K) \end{bmatrix}$$

Each element in f should be a probability for how well input x matches the class.

For an input x, we can define the score as: $\mathbf{w}_k^\top \mathbf{x}$.

2 main problems:
- It can be negative.
- It it not limited between 0 and 1.

To solve this we can easily:

Step 1. Use an exponential to make sure it is always positive.

Step 2. Divide by the whole thing to make sure it is limited between 0 and 1.

$w_k^T x \rightarrow e^{w_k^T x} \rightarrow \frac{e^{w_k^T x}}{\sum_{j=1}^{k} e^{w_j^T x}}$

We get our final output!

$$f(x_i, W) = \frac{1}{\sum_{j=1}^{k} e^{w_j^T x}} \begin{bmatrix} e^{w_1^T x} \\ \cdots \\ e^{w_k^T x} \end{bmatrix} = softmax(Wx)$$

BONUS - For K=2 we recover the Logistic Regression

For K=2, we get the following probability function.

$$f(\mathbf{x};\mathbf{W}) = \begin{bmatrix} \frac{1}{1+\exp\{(\mathbf{w}_1-\mathbf{w}_2)^\top \mathbf{x}\}} \\ 1-\frac{1}{1+\exp\{(\mathbf{w}_1-\mathbf{w}_2)^\top \mathbf{x}\}} \end{bmatrix}$$

And it is reduced to our original assumption (binary probability)

$$P(y=1|x;A;B) = y_{hat}$$
$$P(y=0|x;A;B) = 1-y_{hat}$$

MODEL EVALUATION

- Accuracy:
This is a basic metric that measures the proportion of correctly predicted observations to the total observations.
- Confusion Matrix:
A table used to describe the performance of a classification model.

~~ROC Curve:~~
~~AUC:~~

Only for binary classification!

ASSUMPTIONS OF THE MODEL

1. ~~Binary~~ Multiple Outcome:
The dependent variable ~~is binary~~. can be multiple!

2. No Multicollinearity:
Independent variables should not be highly correlated with each other.

3. Large Sample Size:
Requires a sufficiently large sample size for reliable results.

ML BASICS - MULTIPLE LINEAR REGRESSION EXEMPLIFIED

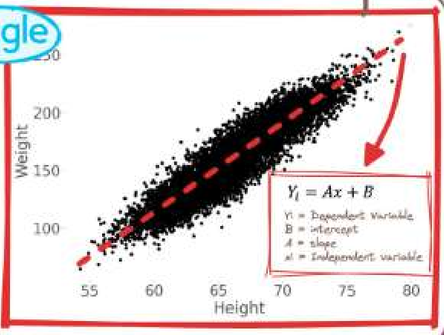
#1 GETTING THE DATA

kaggle

Today we are dealing with some real-world data

And the turn is for... height and weight!

One of the classic examples of linear dependency



However... after a first analysis we observe that

#2 SOME FIRST ANALYSIS

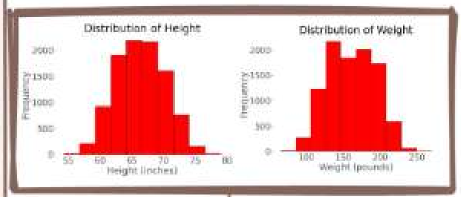
We first check the table

It has 3 main variables:

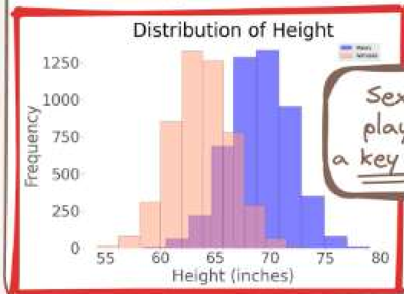
- Sex
- Height
- Weight

	Gender	Height	Weight
0	Male	73.847017	241.893563
1	Male	68.781904	162.310473
2	Male	74.110105	212.740856
3	Male	71.730978	220.042470
4	Male	69.881796	206.349801

We check the distribution of our data



Both variables present a normal distribution



Sex plays a key role

#3 BEYOND HEIGHT - SEX AS A LABEL

Sex plays a key role in humans height and weight proportion

If we split the data set:

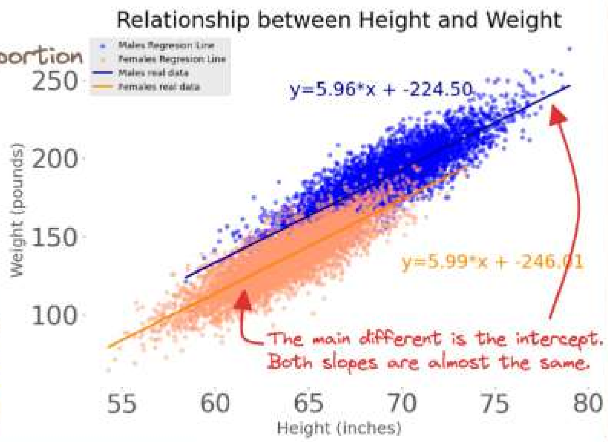
df_males

df_females

And replicate a linear regression for both of them, we obtain two different lines:

y_male = 5.96*x -224.50

y_female = 5.99x - 246.01



#4 MULTIPLE LINEAR REGRESSION

Multiple linear regression uses a linear function to predict the value of a target variable (y)

Containing the function n independent variable $x=[x_1,x_2,x_3,...,x_n]$

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + ... + b_nx_n$$

The weights to be calculated

#5 TYPES OF VARIABLES

Multiple linear regression accepts both numerical and categorical variables

- Numerical variables represent values that can be measured (The height of a person).
- Categorical variables are values that can be sorted in categories (The gender of a person)

To include them in our model, the variable has to be encoded as a binary variable (dummy variable)

	Gender	Height	Weight
0	Male	73.847017	241.893563
1	Male	68.781904	162.310473
2	Male	74.110105	212.740856
3	Male	71.730978	220.042470
4	Male	69.881796	206.349801

pd.get_dummies(df, dtype=int)

	Height	Weight	Gender_Female	Gender_Male
0	73.847017	241.893563	0	1
1	68.781904	162.310473	0	1
2	74.110105	212.740856	0	1
3	71.730978	220.042470	0	1
4	69.881796	206.349801	0	1

df_multiple.drop(...)
df_multiple.rename(...)

	Height	Weight	Gender
0	73.847017	241.893563	0
1	68.781904	162.310473	0
2	74.110105	212.740856	0
3	71.730978	220.042470	0
4	69.881796	206.349801	0

#6 Defining our new equation

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + ... + b_nx_n$$

Starting from the general equation, we just have

One target variable: WEIGHT

Independent variables:
- SEX
- HEIGHT

y_predicted = B + A0 · x_1 + A1 · x_2

Our weights to be determined!

HOW?

1. Gradient Descent (from scratch)
2. OLS (from scratch)
3. Using a pre-build library

#7 Our Predictor and Final Results

```
from sklearn.linear_model import LinearRegression
import random

lr = LinearRegression() # create linear regression object
lr.fit(df_multiple[['Height','Gender']], df['Weight']) # fit linear regression

x_values = np.linspace(55,80,100)
x_values_with_boolean = [(x, random.choice([1, 0])) for x in x_values]

dummy_data = pd.DataFrame(x_values_with_boolean, columns = ["Height","Gender"])
dummy_data["Predicted"] = dummy_data.apply(lambda x:
                x["Height"]*lr.coef_[0] + x["Gender"]*lr.coef_[1] + lr.intercept_, axis=1
                )
```

FINAL RESULT

B = -224.54

A0 = 19.37

A1 = 5.98

Male (x_1 = 0)

Female (x_1 = 1)

$y\_pre = -224.54 + 19.37 \cdot x_1 + 5.97 \cdot x_2$

y_male = 5.98*x -225.55

y_female = 5.98x - 244.92