Outline cho slide nha

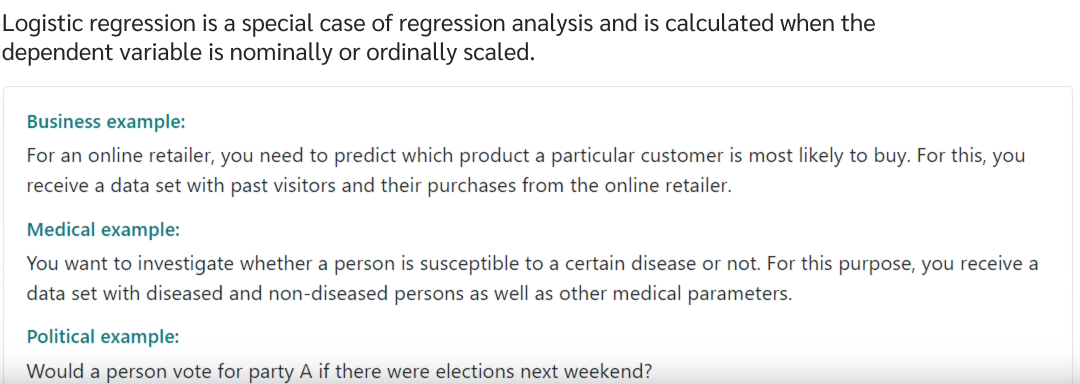
1. **Định nghĩa logistic regression (Thiều)**

What is logistic regression?  
Logistic regression is a process of modeling the probability of a discrete outcome given an input variable.

The most common logistic regression models a binary outcome; something that can take two values such as true/false, yes/no, and so on.

Multinomial logistic regression can model scenarios where there are more than two possible discrete outcomes.

Logistic regression is a type of classification algorithm that is used to predict the probability of a categorical dependent variable based on one or more independent variables.



1. **Example: (Thiều)**

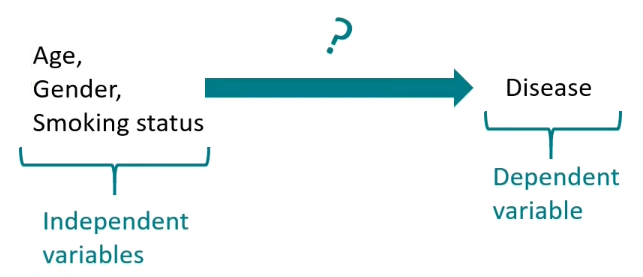
Email sent to your mailbox is spam or not?

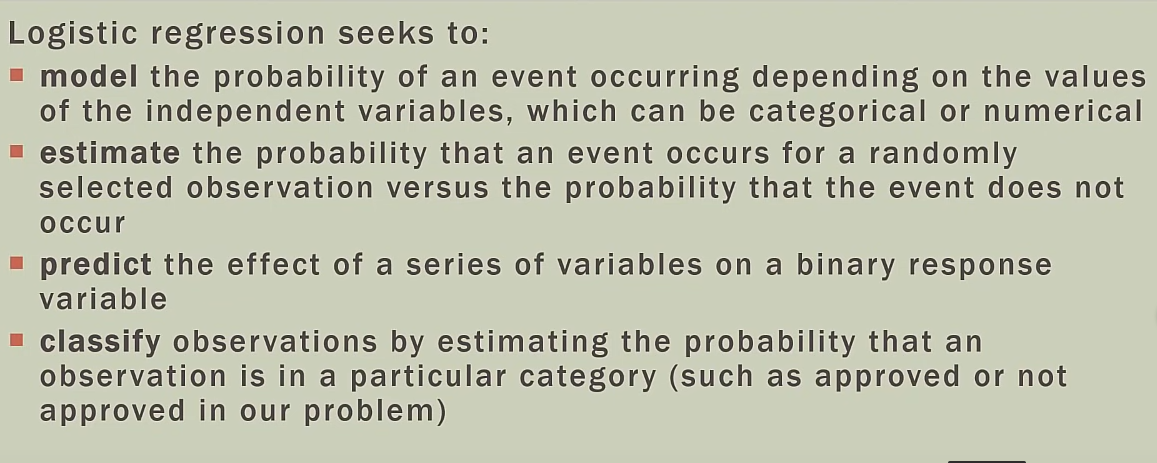
A patient is positive or negative to cancer?

Should I invest in this company or not?

This is a dog or a cat?

* In medicine, for example, a common goal is to find out which variables have an impact on disease.
* In this case, 0 stand for “not diseased” and 1 stand for “diseased”
* And the influence of age, gender, smoking status on this particular disease is examined.





1. **Logistic regression uses the following assumptions:** (Hiếu)

1. The response variable is binary.

It is assumed that the response variable can only take on two possible outcomes.

2. The observations are independent.

It is assumed that the observations in the dataset are independent of each other. That is, the observations should not come from repeated measurements of the same individual or be related to each other in any way.

3. There is no severe multicollinearity among predictor variables.

It is assumed that none of the predictor variables are highly correlated with each other.

4. There are no extreme outliers. (

It is assumed that there are no extreme outliers or influential observations in the dataset.

5. There is a linear relationship between the predictor variables and the logit of the response variable.

6. The sample size is sufficiently large.

As a rule of thumb, you should have a minimum of 10 cases with the least frequent outcome for each explanatory variable. For example, if you have 3 explanatory variables and the expected probability of the least frequent outcome is 0.20, then you should have a sample size of at least (10\*3) / 0.20 = 150.

(Sơn)  
- Định nghĩa learning\_rate, sigmoid function và loss function (vì nó được dùng trong logistic regression (nếu học ATI thì xem lecture 2 của thầy Nam))

**4. Some application of logistic regression (Phương)**Spam detection: Predict whether the mail sent to your mailbox is spam or not.

Credit card fraud: Predict whether a bank transaction is fraudulent.

Health: Predict whether a tumor is negative or positive.

Banking: Predict whether the loan will be repaid or not.

Investment: Predict whether an investment in a start-up will be profitable or not.

**5. Sau đó show code của mình (Hiền, Thảo)**

Topic heart disease predict có thể dùng các techniques khác như: random forest classification, Decision Tree Classifier, KNeighborsClassifier nhưng chọn logistic regression vì:   
- logistic regression is a suitable algorithm for predicting heart disease in Python because it is a powerful and widely used classification algorithm that can model the relationship between predictors and the binary outcome of heart disease.

* Logistic regression is a commonly used classification algorithm in machine learning that is well-suited for binary classification problems where the goal is to predict a binary outcome such as whether a patient has heart disease or not. Logistic regression can model the relationship between the independent variables (predictors) and the dependent variable (outcome) by estimating the probability of the outcome given the predictor variables.
* In the context of predicting heart disease, logistic regression can be used to model the relationship between various risk factors such as age, sex, blood pressure, and cholesterol levels, and the presence or absence of heart disease. By fitting a logistic regression model to historical data, we can learn the relationship between the risk factors and the outcome and use this knowledge to predict the probability of heart disease for new patients based on their risk factor values.

Giải thích code:   
1. Đầu tiên làm như các bước đầu: connect với gg drive => đường dẫn đến file csv để import  
2. Import các thư viện cần thiết

3. The **read\_csv()** function from the pandas library is used to read the CSV file. / Once the CSV file is loaded, the **head()** method is called on the DataFrame object to display the first few rows of data.

4. **data.info()** is a method in pandas library used to display a concise summary of a DataFrame. It gives information about the column names, number of non-null values in each column, data type of each column, and the memory usage of the dataframe.

5. the code **data.isna().sum()** returns the number of missing values in each column of the data dataframe.

(isna() returns a boolean dataframe with the same shape as the original dataframe. Each cell of the boolean dataframe contains True if the corresponding cell in the original dataframe contains a missing value (i.e. NaN), and False otherwise.)  
6. **data.nunique(axis=0)**: This code returns the number of unique values for each column (i.e., axis=0) in the dataset 'data'. (trả về giá trị duy nhất cho một cột nhất định)  
(we can get an idea about the variability and range of values in each column, and this can help in data exploration and understanding the characteristics of the data. Biết được sự biến thiên, đặc điểm của dữ liệu)

7. In this code, the data is cleaned and transformed. The **replace()** function is used to replace the categorical values in the data with numerical values. For example, 'Yes' is replaced with 1 and 'No' is replaced with 0. Similarly, 'Male' is replaced with 1 and 'Female' is replaced with 0. Also, there are some values in the Diabetic column which are not numeric. These values are replaced with 0 and 1 using the **replace()** function.

Finally, the **astype()** function is used to convert the Diabetic column from object type to integer type.

8. **data.HeartDisease.value\_counts()**: output the count of unique values in the 'HeartDisease' column of the dataset.

9. **disease = len(data[ data['HeartDisease'] == 1 ])**

**no\_disease = len(data[ data['HeartDisease'] == 0 ])**

The first line disease = len(data[ data['HeartDisease'] == 1 ]) creates a variable called disease and assigns it the value of the number of rows in the DataFrame data where the 'HeartDisease' column has a value of 1. Here, data['HeartDisease'] == 1 creates a boolean mask, which is used to index the DataFrame data. This will return all the rows where 'HeartDisease' column has a value of 1. The len() function is then used to count the number of rows in the resulting DataFrame.

In summary, these two lines of code are counting the number of individuals in the DataFrame data with and without heart disease and assigning the counts to the variables disease and no\_disease, respectively.

10. import matplotlib.pyplot as plt

plt.rcdefaults()

# Create just a figure and only one subplot

fig, ax = plt.subplots()

y = ('Heart Disease', 'No Disease')

y\_pos = np.arange(len(y))

x = (disease, no\_disease)

ax.barh(y\_pos, x, align='center')

ax.set\_yticks(y\_pos)

ax.set\_yticklabels(y)

ax.set\_xlabel('Count')

ax.set\_title('Target')

for i, val in enumerate(x):

ax.text(val + 10, i, str(val), color='black', va='center', fontweight='normal')

plt.show()

Explain:  
This code is using the Matplotlib library to create a horizontal bar chart to display the number of individuals with and without heart disease in the dataset.

The first line **import matplotlib.pyplot as plt** imports the pyplot module from the Matplotlib library and aliases it as plt.

The second line **plt.rcdefaults()** sets the default Matplotlib parameters.

The next block of code creates a figure with only one subplot and sets the labels and title for the plot.

The y variable is a tuple containing two strings: **'Heart Disease'** and **'No Disease'**, which will be the labels for the bars.

The **y\_pos** variable is a numpy array created using the arange() function, which creates an array of values from 0 to the length of y minus one. This array will be used as the y-axis positions for the bars.

The **x** variable is also a tuple containing the values for the height of each bar, which are the counts of individuals with and without heart disease that were previously calculated and stored in the disease and no\_disease variables.

The **ax.barh(y\_pos, x, align='center')** function creates a horizontal bar chart using the y-axis positions in y\_pos and the bar heights in x. The align parameter centers the bars on the y-axis positions.

The **ax.set\_yticks(y\_pos)** function sets the y-axis ticks at the positions specified in y\_pos.

The **ax.set\_yticklabels(y)** function sets the labels for each tick on the y-axis using the values in y.

The **ax.set\_xlabel('Count')** function sets the label for the x-axis.

The **ax.set\_title('Target')** function sets the title for the plot.

The for loop that follows adds the counts as text labels to the end of each bar.

Finally, **plt.show()** function displays the plot.

11. catcol: This code appears to be creating a list of categorical column names in a dataset.

The variable catcol is a list containing the names of several columns in the dataset that are likely categorical variables.

12. This code is using Matplotlib and Seaborn libraries to create a set of subplots to display the value counts for each categorical column in the dataset. (để tạo một tập hợp các ô con nhằm hiển thị số lượng giá trị cho từng cột phân loại trong tập dữ liệu.)

The first line plt.figure(figsize=(21,42)) sets the figure size to 21 inches by 42 inches.

The for loop iterates through each categorical column in the catcol list. The i variable represents the index of the current column, and the column variable represents the name of the current column.

The plt.subplot(len(catcol), 2, i+1) function creates a subplot grid with the number of rows equal to the length of catcol, two columns, and the index of the current subplot equal to i+1.

The plt.suptitle("Plot Value Count Of Categorical Column", fontsize=20, x=0.5, y=1) function sets the main title for the plot.

The seaborn.countplot(data=data, x=column) function creates a count plot for the current categorical column using Seaborn. The data parameter specifies the dataset, and the x parameter specifies the column to plot.

The plt.title(f"{column}") function sets the title for the current subplot, which is the name of the current categorical column.

The plt.tight\_layout() function adjusts the spacing between the subplots to prevent overlapping.

Overall, this code creates a grid of subplots, with each subplot displaying the value counts for a categorical column in the dataset using Seaborn's countplot() function. The main title at the top of the plot provides an overview of the subplots, and the individual subplot titles indicate which categorical column is being displayed.

13. Numcol : This code appears to be creating a list of numerical column names in a dataset.

The variable numcol is a list containing the names of several columns in the dataset that are likely numerical variables.

14. fig, axes = plt.subplots(2, 2, figsize=(18, 10))

seaborn.histplot(ax=axes[0,0], data=data, x=numcol[0])

seaborn.histplot(ax=axes[0,1], data=data, x=numcol[1])

seaborn.histplot(ax=axes[1,0], data=data, x=numcol[2])

seaborn.histplot(ax=axes[1,1], data=data, x=numcol[3])

This code is using Matplotlib and Seaborn libraries to create a grid of subplots to display histograms of the numerical columns in the dataset.

The fig, axes = plt.subplots(2, 2, figsize=(18, 10)) function creates a 2x2 grid of subplots with a figure size of 18 inches by 10 inches. The fig variable contains the entire figure, and the axes variable is a 2D array of the individual subplots.

The seaborn.histplot(ax=axes[0,0], data=data, x=numcol[0]) function creates a histogram of the first numerical column, 'BMI', and places it in the top-left subplot using the ax parameter. The data parameter specifies the dataset, and the x parameter specifies the column to plot.

The seaborn.histplot(ax=axes[0,1], data=data, x=numcol[1]) function creates a histogram of the second numerical column, 'PhysicalHealth', and places it in the top-right subplot.

The seaborn.histplot(ax=axes[1,0], data=data, x=numcol[2]) function creates a histogram of the third numerical column, 'MentalHealth', and places it in the bottom-left subplot.

The seaborn.histplot(ax=axes[1,1], data=data, x=numcol[3]) function creates a histogram of the fourth numerical column, 'SleepTime', and places it in the bottom-right subplot.

Overall, this code creates a grid of subplots, with each subplot displaying a histogram of a numerical column in the dataset using Seaborn's histplot() function. The ax parameter is used to specify the position of each subplot in the grid, and the x parameter is used to specify the column to plot. The figure size is set to provide adequate space for the subplots.

15. # Since the dataset is undersampled with many data for adults without heart disease, we will oversample the data to make it more balanced

class\_0 = data[data['HeartDisease'] == 0]

class\_1 = data[data['HeartDisease'] == 1]

class\_1 = class\_1.sample( len(class\_0), replace = True )

csv\_df = pd.concat([class\_0, class\_1], axis = 0)

print('Data in data:')

print(data['HeartDisease'].value\_counts())

This code is performing oversampling to balance the class distribution in the dataset.

First, the code creates two separate dataframes, class\_0 and class\_1, by filtering the original data dataframe based on the values in the 'HeartDisease' column. Specifically, class\_0 contains all the rows where 'HeartDisease' is 0, and class\_1 contains all the rows where 'HeartDisease' is 1.

Next, the code uses the sample() function to oversample the minority class, which is class 1. Specifically, the sample() function randomly samples rows from class\_1 with replacement to create a new dataframe of the same length as class\_0. This has the effect of increasing the number of instances of the minority class in the dataset.

Finally, the code concatenates the original class\_0 dataframe and the oversampled class\_1 dataframe using the concat() function, and sets the resulting dataframe to csv\_df. This new dataframe has a balanced class distribution.

The code then prints the value counts of the 'HeartDisease' column in the original data dataframe to show the imbalance, but does not print the value counts of the balanced csv\_df dataframe.

16. This code is splitting the dataset into training and testing sets using the train\_test\_split() function from the sklearn.model\_selection module.

First, the code creates two dataframes: features and target. The features dataframe contains all the columns in the original data dataframe except for the 'HeartDisease' column, which is the target variable. The target dataframe contains only the 'HeartDisease' column.

Next, the code calls train\_test\_split() function and passes in the features and target dataframes, as well as some additional arguments. Specifically, test\_size=0.2 specifies that the testing set should contain 20% of the data, and random\_state=0 sets the random seed for reproducibility.

The function returns four variables: X\_train, X\_test, y\_train, and y\_test. X\_train and y\_train contain the training set, while X\_test and y\_test contain the testing set.

Finally, the code prints the shapes of the features and target dataframes, as well as the shapes of the training and testing sets.

17. This code is performing one-hot encoding on the categorical features 'AgeCategory', 'Race', and 'GenHealth' in both the training and test datasets.

The make\_column\_transformer function from the sklearn.compose module is used to define the transformation pipeline. The pipeline applies OneHotEncoder to the categorical columns and passes through the remaining columns without any changes.

The training and test datasets are transformed using the fit\_transform method of the transformer object. The resulting transformed data is stored in dataframes transformed\_train\_data and transformed\_test\_data, respectively. The new columns generated by one-hot encoding are given appropriate names using get\_feature\_names\_out() method of the transformer object.

The transformed data is concatenated with the original dataframes using pd.concat(). The old columns that were one-hot encoded are then dropped from the dataframes using the drop() method with the axis parameter set to 1.

The resulting transformed training and test datasets are stored in X\_train and X\_test variables, respectively.

18. The code performs data standardization using the StandardScaler class from the scikit-learn library.

In machine learning, it is often useful to standardize the data, which means to transform it so that it has a mean of 0 and a standard deviation of 1. This can help the machine learning algorithm to converge faster and may improve the accuracy of the model.

The StandardScaler class is used to perform this standardization on the data. The fit\_transform() method is called on the X\_train data to fit the scaler to the data and then transform it. The same scaler is then used to transform the X\_test data using the transform() method.

19. Logistic regression

The code above fits a logistic regression model to the preprocessed and standardized data using the LogisticRegression class from scikit-learn. The max\_iter parameter is set to 1000 to ensure convergence of the optimization algorithm. The model is then used to make predictions on the test set, and the resulting predictions are stored in a pandas DataFrame called output.

The code then calculates the percentage of people in the test set who are predicted to have heart disease by looking at the number of predicted positive cases and dividing by the total number of cases. Finally, the classification\_report function from scikit-learn is used to generate a report of various classification metrics, such as precision, recall, and F1 score, for the model's performance on the test set.

20. Confusion matrix

The confusion matrix shows the performance of the logistic regression model on the test set. It indicates the number of true positive (predicted as positive and actually positive), false positive (predicted as positive but actually negative), true negative (predicted as negative and actually negative), and false negative (predicted as negative but actually positive) predictions.

In this case, the model correctly predicted 28 true positive cases and 23 true negative cases, while it made 2 false positive predictions and 1 false negative prediction. Overall, the model performed relatively well, but there is still room for improvement.

21.

The ROC curve (Receiver Operating Characteristic curve)

is a graphical representation of the performance of a binary classification model, showing the trade-off between the true positive rate (sensitivity) and false positive rate (1-specificity) at different classification thresholds. The AUC (Area Under the Curve) is a measure of the overall performance of a binary classification model based on the ROC curve, where an AUC of 1 indicates perfect performance, while an AUC of 0.5 indicates a model with no predictive power.

In this case, the AUC for the logistic regression model is 0.89, which indicates a good performance in distinguishing between individuals with and without heart disease. The ROC curve shows a steep increase in the true positive rate for a low false positive rate, which suggests that the model has good discriminatory power.