**Machine Learning for Classification**

Classification is usually the next task in the process of learning about machine learning after linear regression. Classification refers to a group of machine learning problems where instead of having a continuous target variable, we have a target which can usually have countable unique values and **counts usually lie below 10**. It goes beyond 10 in very rare problems.

Classification problems can further be classified into the following two types:

1. Binary classification problems
2. Multi-class classification problems

Please note that we are only talking about problems (and/or) tasks here not the actual machine learning algorithms which are used to solve these.

Let us briefly list down all the algorithms which are out there to solve a machine learning task:

1. Logistic Regression
2. Support Vector Machines
3. Naive Bayes algorithm
4. Decision Tree
5. Random Forest
6. Neural Networks

In the last chapter, we discussed the Linear Regression algorithm (which is a Supervised ML algorithm) where the objective was to predict a continuous target variable. Here, as an analogy of the last chapter, we will start the discussion with classification algorithms (especially the Logistic Regression algorithm) for solving these problems.

**CRISP - DM PROCESS**

Let us review this process once again, as this process is followed whenever we are trying to use machine learning to solve any problem. We will follow the CRISP-DM process of machine learning in this chapter to do a guided project on predicting stroke probability which is a binary classification problem where we are trying to predict whether an individual is likely to get a stroke or not.Here are the six steps to do machine learning using **CRISP-DM**:

1. **Business Understanding**
2. **Data Understanding**
3. **Data Preparation and Exploratory data analysis**
4. **Modeling**
5. **Evaluation and Validation**
6. **Deployment**

Let us get started.

**Objective of this chapter**

Before we proceed, open [**this**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD)link. This link will open an **online hosted Jupyter Lab environment that** can be used to run the codes discussed in this lab and also try the exercises. You might have to wait a few minutes for it to fully load. This will have the labs, exercises and project notebooks of all the chapters in this book.

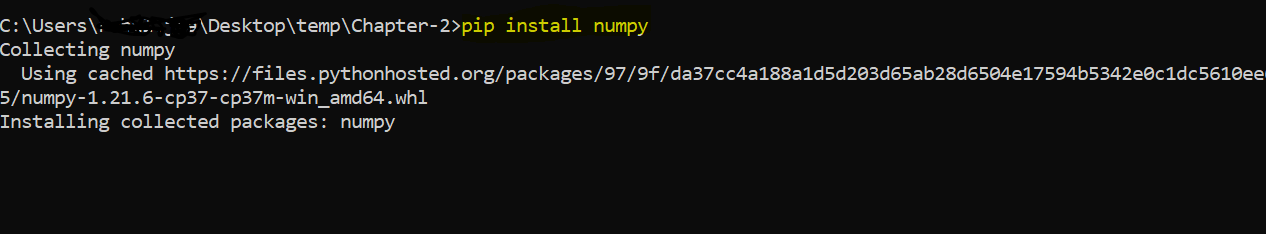
Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

We will do a guided project to predict whether an individual is likely to get a stroke or not in future based on various features about that individual and this data This data can be viewed [**here**](https://www.kaggle.com/fedesoriano/stroke-prediction-dataset). It is also made available on the online hosted environment which we provided for you. So, you don’t need to download this data as everything is available on the online hosted lab.

Data Link: **https://www.kaggle.com/fedesoriano/stroke-prediction-dataset**

We will be using the Python programming language to do this guided project and the libraries you will need are “Numpy”, “Skearn” and “Pandas”. **You don’t need to install anything as we will be providing you with a hosted notebook on binder which you can run to execute codes as we move further into this chapter**.

But if you are working on your local system, then make sure that you have numpy installed on your system. If it is not already installed, open a command prompt and enter the following command to install numpy:



**Machine Learning Process**

As discussed, We will follow the **CRISP-DM** steps to do the project on **cancer\_rate** prediction.

**Step.1 Business Understanding**

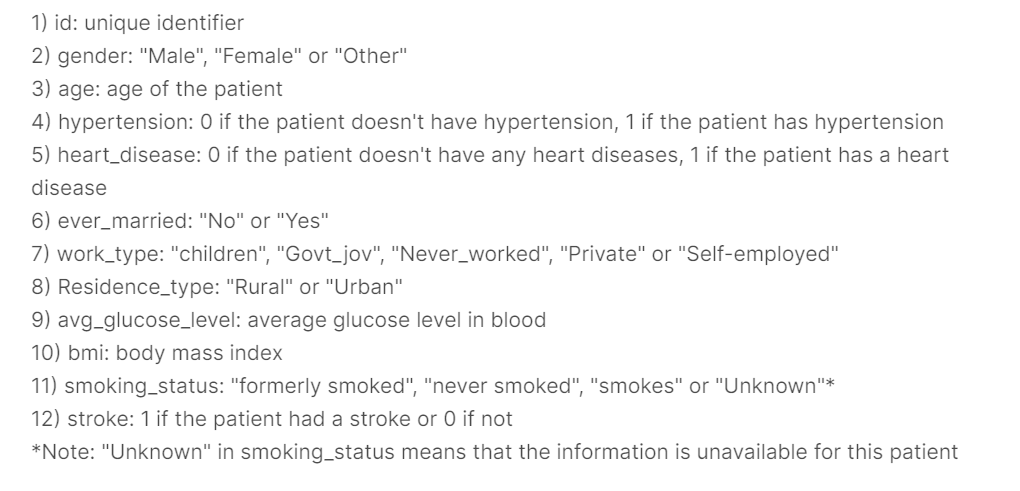
The World Health Organization (WHO) states that stroke is the one of the leading causes of death globally, responsible for **approximately 11% of total deaths**. So, it would be desirable to build a model which can predict a patient’s stroke likelihood accurately.

We will be doing a guided project in this lesson on predicting whether an individual can have a stroke or not in future based on the individual's characteristics. If this model is built correctly with good accuracy, it may add to the knowledge of doctors as they might use this model to further enhance their understanding about the person’s likelihood of getting a stroke. If we can explain the model features using SHAPE values, then for each individual we can get information on which feature is contributing to the predicted likelihood of stroke which doctors can then use with their current understanding of the patient's health.

This model can also be used by government agencies to further the spending on the analytics in the research space.

**Step.2 Data Understanding**

The data come from the online data science competition platform kaggle and it has the following features:



As we can see, this represents various patient’s health related characteristics which might be determinant of stroke further some possibly stress related variables (like type of work, married, hypertension Etc.)

**Step.3 Data Preparation and Exploratory data analysis**

Once we are done with the initial two phases, which in real setting, can even take days to complete, we can get started with the data preparation part which involves joining tables from various data warehouses. Once the data preparation is done, we need to understand the data as much as possible by doing the exploratory data analysis. This step is kind of an art where you need to ask as many questions as possible.

There are no strict rules when it comes to Exploration of your data, you can do it your own way but the main point of doing it is to become as much as familiar with your data because this would help in choosing features which might be needed in the modeling step.

**Objective of Exploration**

Objective of the exploration is to become acquainted with your data and try to look at it from each and every angle as possible. The key to best EDA is to :

1. Ask as many questions as possible about your data.
2. Try to answer those using your exploration

If you want to do your exploration in python, then you need to be good in pandas to do these two steps effectively.

**Let us look at some of the methods of exploration of this data.**

Although this is not an exhaustive list, some of the initial steps of any exploration is to get a quick feel of the data we have and it might involve answering the following questions:

1. *Number of rows and columns of the data*
2. *Number of categorical and continuous variables*
3. *Separating the target variable*
4. *Correlation plots of target vs numeric variable*
5. *Categorical data vs target (Looking at the group by average)*

Most of the above points were also there in the previous chapter, we will also look at some of the additional steps of exploration such as:

1. Feature importances of numerical and categorical variables
2. Categorical variable encoding (To feed them into our model)
3. Missing value imputation

So, by the end of this exploration, we might have a good understanding of the stroke data that we are using. Further, we might have a set of good features which can be used in the logistic regression model. So, let us get started.

So, let us look at the above exploration steps one by one. After the end of these steps, we will have an exercise, where you will get a chance to run these code on the online hosted environment.

**Reading the data and looking the first few rows**

First step of any exploratory analysis is to get the data into python and then using pandas functions, it can be analyzed.

This is how we read python data\_frame which are usually in CSV format:

**import pandas as pd**

**data = pd.read\_csv(<PLEASE SPECIFY A PATH HERE(WHERE YOUR DATA RESIDED)>)**

Once you specify the data path, a pandas data\_frame object will be stored in the “data” variable and there are two questions which can be answered using this variable:

1. View the shape of the data which will give us the idea about number of rows and columns
2. Also, we can look at the first few rows of the data to get a quick glance of the data.

We can call the following attribute for checking the head:

**data.head()**

The shape can be checked using the following code:

**data.shape**

**Missing Value analysis:**

We will be using the sklearn library for Logistic Regression modeling. This library expects a data frame in either numpy array format or in the pandas data frame but we can’t have missing values in the data frame. We have the following two options to deal with them:

1. Remove the rows containing missing values.
2. Remove the columns containing missing values
3. Impute these missing values.

Both option 1 and option 2 lead us to a situation which is a data loss and in a situation where we have the lack of data, we can’t do that. So, we will adopt the third strategy in this lesson. In the last chapter, we removed the columns which had missing values and hence we adopted the 2nd strategy there.

**Missing Values**

Let us view how many missing values each of our data have. This is important because the modeling step cannot be executed without treating the missing values.

Missing values can be treated either by just dropping them or imputing those with some technique. Dropping does not always make sense if the number of missing values are quite large.

We can look at missing values in any pandas dataframe using the following code:

**pd.isnull(<data\_name>).sum()**

**Missing value treatment**

There can be two situations here:

**Missing values in numeric variable**

These can be imputed by some simple strategy such as mean,median or mode imputation. To know which strategy to use, we will be checking the distribution of the numeric variable.

To check the distribution of the data, we use the “.hist” method on the column of data frame to get the histogram plot and from there, we can make the decision based on the following criteria:

1. If the distribution is symmetric (normal), we can use any of these measures, it doesn't matter
2. If the distribution is a right or left tail, then we can prefer median over mean but business knowledge might guide the decision better here.
3. Mode is generally preferred when the distribution has few distinct values. But mean and median should also just work fine in those situations too.

**Missing values in categorical variable**

These are usually imputed by mode strategies as other strategies do not make sense for a categorical variable.

**One pandas function to do both of those**

The fillna method on a data frame can handle both the cases but you need to know the value which needs to be imputed.

**<data\_name>[<column\_name>].fillna(value\_that\_need\_to\_be\_imputed)**

The value can be known by :

1. First decide the approach i.e mean, median or mode.
2. Then call this method on the column which contains the missing values.

For the mean:

**<data\_name>["column\_name"].mean()**

For the median:

**<data\_name>["column\_name"].median()**

For the mode:

**<data\_name>["column\_name"].mode()**

**Numeric columns and Categorical Columns**

The columns of a dataframe can be printed using the .columns attribute of the pandas dataframe.

There is a function in the pandas called “select\_dtypes” which makes it easy to subset the numeric columns and categorical columns for us.

In this function, we can pass the value of include as “object” to get all the categorical variables like this:

**<dataframe-name-here>.select\_dtypes(include="object").columns**

If on the other hand, you pass the value as “number”, you will get all the numeric column names.

**<dataframe-name-here>.select\_dtypes(include="number").columns**

**Exercise. 1.1**

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Open the Exercise 1.1 file from the lab [hosted here](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD) (navigate to chapter 3) and do the following:

1. Read the data using pandas:

**import pandas as pd**

**data = pd.read\_csv(r"https://raw.githubusercontent.com/fenago/MLWorkshop/main/Chapter-3/stroke.csv")**

1. Check the shape:

**data.shape**

So, we have around 12 columns out of which one is “id” and another is target. So, the useful features are only 10.

1. Let us look at the top 10 rows:

**data.head()**

1. It will show us some basic info but still it does not give us the information on the data types, Let us check that:

**data.info()**

1. It does give us the data types of each column but what if we want to store this info in a separate variable? Let us do that using:

**num\_cols = data.select\_dtypes(include="number").columns**

**num\_cols\_rem = num\_cols[~num\_cols.isin(["id", "stroke"])]**

**categorical\_cols = data.select\_dtypes(include="object").columns**

Here, we have just extracted the numeric solemn first and removed the “id” and “stroke” variable from this data.

1. Check which columns have missing values:

**pd.isnull(data).sum()**

So, only bmi (Which is a numeric variable) has missing values. So, we can fill them with the mean or median.

1. Look at the distribution of the BMI column:

**data.bmi.hist()**

1. The column is not perfectly normal, so we can use median to fill in the missing value:

**data.bmi.fillna(data.bmi.median(), inplace=True)**

Inplace is required because you want original data to be modified.

**Categorical Encoding**

Categorical encoding, as the name suggests, is a way to convert the categorical data type to numerical data type, so that the sklearn can process these. Mainly, there are two ways to do this encoding:

1. Ordinal Encoding: Which just replaces all categories by numbers from 1 to number of unique categories in that variable
2. One Hot Encoding: For every column, we create a **number\_of\_unique\_categ - 1** new column where each column will have 1 where the ith category was present and 0 all other places.

So, if I have a column called gender which may contain three values “Male”, “Female” and “Other”. Here, there are 3 **number\_of\_unique\_categ.So, we will create 2 new columns.** First will be **gender\_male** and **gender\_female**. We don’t create gender\_other because wherever both gender\_male and gender\_female are zero, that would be understood as the third gender and hence doing it again will lead to creating extra variables.

**Implementing encoding in Python**

So, there are following two steps for encoding any categorical column in sklearn (irrespective of the type of encoding being applied here):

1. Fit the encoding to the data (Usually the training data).
2. Transform the input data using the fitted encoder (usually both training and testing data).

Let us look at the pseudocode for these two steps (usually you want to have train and test split made before doing this step, we discussed this step in the last chapter):

Also, this step assumes that you have taken care of missing values from your data.

Importing Packages:

**from sklearn.preprocessing import OrdinalEncoder,OneHotEncoder**

**from sklearn.model\_selection import train\_test\_split**

Initializing the class for the appropriate encoder:

**scale = OrdinalEncoder(handle\_unknown="use\_encoded\_value", unknown\_value = -1)**

**ohe = OneHotEncoder(handle\_unknown="ignore")**

Finally, fit these encoders on the train test:

**X = data.drop(columns=["id", "stroke"])**

**y = data["stroke"]**

**X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.2,stratify=y)**

**scaler.fit(X\_train)**

**ohe.fit(X\_train)**

So, we are doing the following:

1. Drop id and target variable from the X matrix
2. Store target into y
3. Make a train and test split
4. Use the train test to fit both ordinal and one hot encoders.

Handle\_unknown parameter

This parameter was passed in the class initialization to tell the class object what method will be used if any unknown categories are found during the transform step. What does this mean?

So, when we fit our encoder to a data, it will only remember the categories present in it. This handle\_unknown parameter tells the encoder about the treatment when a new category is found in the transform step.

Let us take an example.

Suppose, we had a training set with a gender column which had only two values “Male” and ”Female”. So, our ordinal encoding will learn these two values during the fit step and it will replace “Male” with 1 and “Female” with 0 but what should it do if it finds some new category in the transform step, say “Other”. In that case, we are also providing unknown\_value = -1, so the encoder will code -1 to every unknown value it finds in the transform step.

So, finally let us look at the code to transform the data from categorical to numerical.

**X\_train\_transform\_ord = scaler.transform(X\_train)**

**X\_test\_transform\_ord = scaler.transform(X\_test)**

**X\_train\_transform\_ohe = ohe.transform(X\_train)**

**X\_test\_transform\_ohe = ohe.transform(X\_test)**

If you check the shape of all X’s, then you will notice that ordinal encoding does not change the number of colulums while one hot encoding does.

**Exercise. 1.2**

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Open the Exercise 1.2 file from the lab [**hosted here**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD)(navigate to chapter 3) and do the following:

1. The code from the previous exercise will be available for you to create the data and all. Run whatever is there in the file.
2. Train and Test split the data, since we have to only encode categorical data, we will split the data into two different data matrices, one for numerical and another for categorical. Then, they will be merged later once our encoding is done. We will be using the two variables we created in the previous exercise (Which hold the numerical and categorical column):

**from sklearn.model\_selection import train\_test\_split**

**X = data.drop(columns=["id", "stroke"])**

**y = data["stroke"]**

**X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.2,stratify=y)**

**X\_train\_cate = X\_train[categorical\_cols]**

**X\_test\_cate = X\_test[categorical\_cols]**

**X\_train\_num = X\_train[num\_cols\_rem]**

**X\_test\_num = X\_test[num\_cols\_rem]**

1. Instantiate the class for ordinal and one hot encoders:

**from sklearn.preprocessing import OrdinalEncoder,OneHotEncoder**

**scaler = OrdinalEncoder(handle\_unknown="use\_encoded\_value", unknown\_value = -1)**

**ohe = OneHotEncoder(handle\_unknown="ignore")**

1. Fit then on the X matrix for train and which contain only categorical data:

**scaler.fit(X\_train\_cate)**

**ohe.fit(X\_train\_cate)**

1. Transform the train and test using the adobe two encoder, we wil have four different data set:

**X\_train\_transform\_ord = scaler.transform(X\_train\_cate)**

**X\_test\_transform\_ord = scaler.transform(X\_test\_cate)**

**X\_train\_transform\_ohe = ohe.transform(X\_train\_cate)**

**X\_test\_transform\_ohe = ohe.transform(X\_test\_cate)**

1. Finally, merge the numerical columns back to the categorical ones:

**X\_train\_transform\_ord = pd.DataFrame(X\_train\_transform\_ord, columns=scaler.feature\_names\_in\_)**

**X\_test\_transform\_ord = pd.DataFrame(X\_test\_transform\_ord, columns=scaler.feature\_names\_in\_)**

**X\_train\_transform\_ohe = pd.DataFrame(X\_train\_transform\_ohe.toarray(), columns=ohe.get\_feature\_names\_out())**

**X\_test\_transform\_ohe = pd.DataFrame(X\_test\_transform\_ohe.toarray(), columns=ohe.get\_feature\_names\_out())**

**X\_train\_num.reset\_index(inplace=True,drop=True)**

**X\_test\_num.reset\_index(inplace=True,drop=True)**

**X\_train\_ord = pd.concat([X\_train\_transform\_ord, X\_train\_num],axis = 1)**

**X\_test\_ord = pd.concat([X\_test\_transform\_ord, X\_test\_num],axis = 1)**

**X\_train\_ohe = pd.concat([X\_train\_transform\_ohe, X\_train\_num],axis = 1)**

**X\_test\_ohe = pd.concat([X\_test\_transform\_ohe, X\_test\_num],axis = 1)**

**Feature Importances**

Our target variable is **stroke** which is numeric and can only take two distinct values such as 0 and 1. We would like to know which features are best related to these “stroke” variables.

In the last chapter, to achieve this, we took the help of correlation and it was fine because we only dealt with numeric variables but in this chapter, we need to include categorical variables also. Now, that we have converted the categorical variables into numeric, we could in theory use the previous method to get the feature importances but it is not a very convenient function, so we will use a sklearn’s mutual information score to get these scores.

The following code can be used to get the top features for our model.

**from sklearn.feature\_selection import mutual\_info\_classif**

**values = mutual\_info\_classif(X\_train\_ord, y\_train)**

This will just give us a bunch of value, the higher the value, the better the feature. We can get the column names and match up what are the top features of our model.

**Exercise. 1.3**

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Open the Exercise 1.3 file from the lab [**hosted here**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD)(navigate to chapter 3) and do the following:

1. Run the code given already in the notebook.
2. Get the top feature values by the following code:

**from sklearn.feature\_selection import mutual\_info\_classif**

**values = mutual\_info\_classif(X\_train\_ord, y\_train)**

1. Run a loop which will iterate through these values and get you the column names and corresponding feature importance value:

**for i, col in enumerate(X\_train\_ord.columns):**

**print(col, " has the feature importance = ", values[i])**

1. Now, this can be used to even get top values:

**values\_dict = {col:values[i] for i,col in enumerate(X\_train\_ord.columns)}**

**sorted(values\_dict,key=lambda x:x[1])**

**Modeling**

We have two sets of train and test splits for two different methods of encoding (i.e Ordinal and One hot encoding). By the end of this chapter, we will try to evaluate both of these methods using accuracy (which is one of the metrics of classification algorithms).

But before that, let us get started with the modeling procedure.

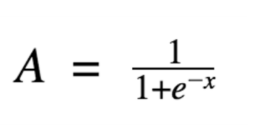
We will be looking at one of the most simplest yet powerful algorithms for doing classification tasks in ML and that algorithm is called Logistic Regression.

**From Linear Regression to Logistic Regression**

The problem that logistic regression is trying to solve requires a binary kind of prediction. If you remember, in linear regression, we had a linear combination of features and if weights (coefficients) become really large, then it may make the value of the target really huge. Suppose, if corresponding to one row of patients (Who had the stroke), we get as a result of linear regression a value which is 800. How do we interpret this value? It is very difficult to interpret this value and this is the reason we don’t use linear regression for classification tasks where the target can only take two possible values 0 and 1.

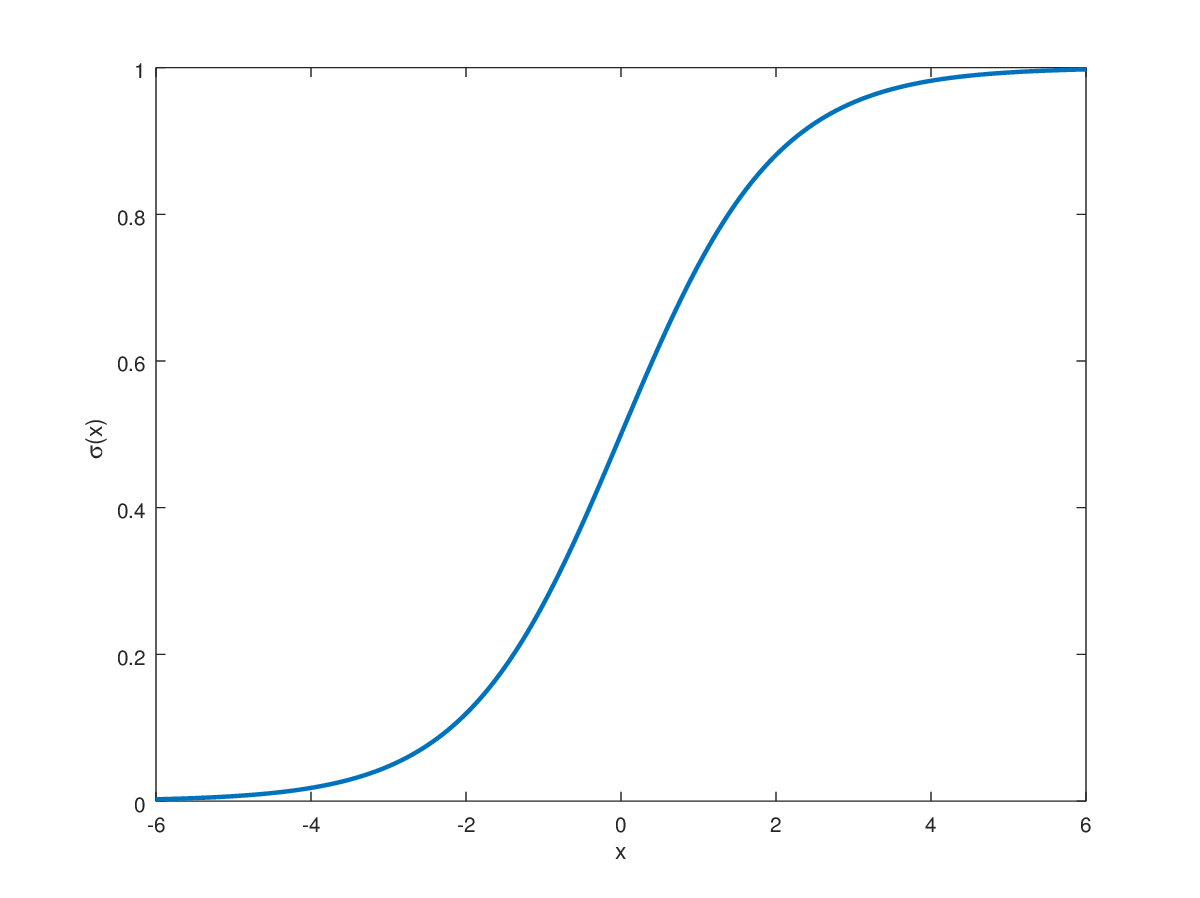
So, we need to somehow scale the value given by linear regression so that those values make sense. What we do is, we take a sigmoid transformation of the linear regression out and get a value which is between 0 and 1. In this way, we can interrupt this number as a probability of a stroke. The closer it is to 1, the better it is.

This is how we do this:



The x here is the value calculated by usual linear regression. The value of A will be between 0 and 1.

Hence, it can be interpreted as the probability of a stroke. The graph of sigmoid as a function of x looks like the following curve:



The higher the value of x, the larger the probability will be, hence when we said that if the value is 800, how do we interpret it. So, here is the answer. This value might mean that the predicted target would be 1. Because it will lie to the extreme right of the above plot.

**Logistic Regression in Sklearn**

To build a model in sklearn, we will again follow the four step procedure we discussed in the last lecture.

Now, the fun part. The Sklearn library makes it easy for us to implement ML algorithms.

We already have our train and test setup and just need to implement the sklearn part. Any model is trained in sklearn using the following steps:

1. Import the required model class from the package, here we need to import LinearRegression from sklearn.linear\_model

**from sub\_package import Model\_class**

1. Initialize the class by saying:

**model\_name = Model\_class()**

1. Call the fit method to train the model:

**model\_name.fit(X\_train, y\_train)**

1. To make the predictions:

**predictions\_test = model\_name.predict(X\_test)**

Yes, it was this simple. We can fit and predict using any model from sklearn using just the above four steps.

**Exercise. 1.4**

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Open the Exercise 1.4 file from the lab [**hosted here**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD)(navigate to chapter 4) and do the following. This will have the train and test splits made for you and also it will have the above implementation ready to use.

1. Run all the cells
2. Use four steps procedure on X\_train\_ord and y\_train arrays

**from sklearn.linear\_model import LogisticRegression**

**model\_ord = LogisticRegression(max\_iter=1000)**

**model\_ord.fit(X\_train\_ord, y\_train)**

1. Repeat the same procedure on the One hot data matrix:

**model\_ohe = LogisticRegression(max\_iter=1000)**

**model\_ohe.fit(X\_train\_ohe,y\_train)**

1. **Store all the predictions from both the models in train and test data:**

**train\_preds\_ord = model\_ord.predict(X\_train\_ord)**

**test\_preds\_ord = model\_ord.predict(X\_test\_ord)**

**train\_preds\_ohe = model\_ohe.predict(X\_train\_ohe)**

**test\_preds\_ohe = model\_ohe.predict(X\_test\_ohe)**

**Evaluation**

We have trained our model, predicted on train and tested using the trained model but how does our model perform? Also, we predicted using our model for two different sets of data (for ordinal and one hot encoding respectively)

To answer that, we need to look at evaluation metrics. Let us recap the meaning of metrics in machine learning

**What is the metric?**

Metric is one single value which tells you about the performance of your model. By itself, it has no worth but when you have two metric values, both can be compared. So, metrics are useful only when they are available for many algorithms so that all of those algorithms can be compared.

Metrics can be different for different ML tasks. There can be two ML tasks:

1. **Regression**
2. **Classification**

Since in this chapter the task is classification, let us continue with that.

In binary classification, our basic objective is to predict the probability of a stroke as close as possible to 1. How could we possibly create a metric here?

One way is thresholding. This way, we can create a binary prediction column which will be based on the threshold value. Say if prob > 0.5, we will predict 1 and otherwise we will predict 0.

Now, what we can do is: take the proportions of correct predictions made by our model as one of the metrics. This is just the simplest one. In the next chapter, we will discuss the metrics in machine learning models in more detail. Some of the other classification metrics are:

1. Precision
2. Recall
3. F1 Score
4. AUC ROC
5. Etc.

We will discuss these in more detail in the next chapter, but let's implement accuracy using numpy.

**Accuracy using numpy**

It is one of the easiest metrics to implement in sklearn. The following steps can be followed to get the value of accuracy using numpy:

1. Get the prediction using the model
2. Set the threshold using if else statement
3. Make a element wise comparior and track the counts where they are equal
4. Divide the count by the length of the dataset.

**Exercise. 1.5**

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Open the Exercise 1.5 file from the lab [**hosted here**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD)(navigate to chapter 3) and do the following. The train and test predictions have been provided to you.

1. Run all the cells with code
2. Implement the accuracy function :

**def acuarcy\_fun(true, preds):**

**counts = sum(true == preds)**

**return counts/len(true)**

1. Test this function on one hot encoding test data:

**accuracy\_score(y\_test, test\_preds\_ohe)**

**Accuracy using sklearn**

As you might have guessed, sklearn also has functions for computing various metric values.

It can be imported as:

**from sklearn.metrics import accuracy\_score**

**Exercise. 1.6**

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Open the Exercise 1.6 file from the lab [**hosted here**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD)(navigate to chapter 3) and do the following. The train and test predictions have been provided to you.

1. Run all the cells with code
2. Import the accuracy measure from sklearn:

**from sklearn.metrics import accuracy\_score**

1. Compute accuracy on ordinal train and test:

**print("Train acc: (Ordinal Encoding)",accuracy\_score(y\_train, train\_preds\_ord))**

**print("Test acc: (Ordinal Encoding)",accuracy\_score(y\_test, test\_preds\_ord))**

1. Compute accuracy on one hot train and test:

**print("Train acc: (One Hot Encoding)",accuracy\_score(y\_train, train\_preds\_ohe))**

**print("Test acc: (One Hot Encoding)",accuracy\_score(y\_test, test\_preds\_ohe))**

1. Which Encoding type is better?

One one seems to be performing at a similar level but one hot encoded works slightly better here.

**Deployment**

Finally, if you are satisfied with the evaluation values you are getting in the previous step and trying all the different algorithms, you can finalize the model. After finalizing, we can take one of the following steps:

1. Share the predictions with the appropriate team, so that they can take appropriate action based on the predictions.
2. Create a web based framework to serve our predictions to users

**Activity**

Open the Activity 1.ipynb file from the lab [hosted here](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD) (navigate to chapter 3) and do the following. The objective is to predict water portability based on various feature .

1. The initial starter code is available. Explore the data using the steps we discussed in this chapter.
2. Treat missing values by using the mean/median strategy for the numeric variables and for the categorical use the mode strategy
3. Include all the categorical variables by doing one hot encoding. Use the handle\_unknown as ignore
4. Create a train and test split
5. Compute the feature importances and use the top 10 variables for the modeling
6. Create a logistic regression model using sklearn API
7. Evaluate the model using an accuracy measure.

**Project**

Navigate to the kaggle link [**here**](https://www.kaggle.com/datasets/pavansubhasht/ibm-hr-analytics-attrition-dataset) and build a classification model to predict employee attrition. Try to include all the steps that we followed in this chapter including the missing value treatment and including categorical variables in your model.

**Summary**

In this chapter, we discussed machine learning for classification. We covered how logistic regression is just a non-linear extension of Linear Regression. Further, we looked at how to interpret the predictions of Logistic Regression. We can interpret these as just the probability of the positive class (I.e likelihood of stroke).

We also looked at how to deal with missing values for both numeric and categorical variables.

Further, we looked at how to include categorical variables in our machine learning model. We studied two different types of encoding methods called ordinal encoding and one hot encoding.

Finally, we studied accuracy as a metric score for the classification problems and we compared the results of metrics for both ordinal and one hot encoding data matrix.