|  |  |
| --- | --- |
| Name Of The Student | Himanshu |
| Internship Project Topic | TCS iON RIO-210: Build a Classification Model for Drug Trials Dataset |
| Name of the Organization | TCS iON |
| Name of the Industry Mentor | Himdweep Walia |
| Name of the Institute | Amity University |

|  |  |  |
| --- | --- | --- |
| Date | Day # | Hours Spent |
| 22-05-2024 | Day-29 | 5 Hours |
| **Learn about the accuracy checking of Machine learning models.**  To implement a simple machine learning algorithm in Python using Scikit-learn :  **Step 1 — Importing Scikit-learn**  Make sure you’re in the directory where the environment is located, and run the following command:   |  | | --- | | $ . my\_env/bin/activate |   To install sklearn , and run the following command:   |  | | --- | | $ pip install scikit-learn |   Once the installation completes, launch Jupyter Notebook:   |  | | --- | | $ jupyter notebook |   **Step 2 — Importing Scikit-learn’s Dataset**  The dataset we will be working with in this tutorial is the Breast Cancer Wisconsin Diagnostic Database. The dataset includes various information about breast cancer tumors, as well as classification labels of malignant or benign. The dataset has 569 instances, or data, on 569 tumors and includes information on 30 attributes, or features, such as the radius of the tumor, texture, smoothness, and area.  Scikit-learn comes installed with various datasets which we can load into Python, and the dataset we want is included. Import and load the dataset:   |  | | --- | | import sklearn  from sklearn.datasets import load\_breast\_cancer  # Load dataset  data = load\_breast\_cancer() |   The data variable represents a Python object that works like a dictionary. The important dictionary keys to consider are the classification label names (target\_names), the actual labels (target), the attribute/feature names (feature\_names), and the attributes (data).  Attributes are a critical part of any classifier. Attributes capture important characteristics about the nature of the data. Given the label we are trying to predict (malignant versus benign tumor), possible useful attributes include the size, radius, and texture of the tumor.  Create new variables for each important set of information and assign the data:   |  | | --- | | # Organize our data  label\_names = data['target\_names']  labels = data['target']  feature\_names = data['feature\_names']  features = data['data'] |   We now have lists for each set of information. To get a better understanding of our dataset, let’s take a look at our data by printing our class labels, the first data instance’s label, our feature names, and the feature values for the first data instance:   |  | | --- | | # Look at our data  print(label\_names)  print(labels[0])  print(feature\_names[0])  print(features[0]) |   IMG_256  As the image shows, our class names are malignant and benign, which are then mapped to binary values of 0 and 1, where 0 represents malignant tumors and 1 represents benign tumors. Therefore, our first data instance is a malignant tumor whose mean radius is 1.79900000e+01.  Now that we have our data loaded, we can work with our data to build our machine learning classifier.  To evaluate how well a classifier is performing, you should always test the model on unseen data. Therefore, before building a model, split your data into two parts: a training set and a test set.  You use the training set to train and evaluate the model during the development stage. You then use the trained model to make predictions on the unseen test set. This approach gives you a sense of the model’s performance and robustness.  Fortunately, sklearn has a function called train\_test\_split(), which divides your data into these sets. Import the function and then use it to split the data:   |  | | --- | | from sklearn.model\_selection import train\_test\_split  # Split our data  train, test, train\_labels, test\_labels = train\_test\_split(features, labels, test\_size=0.33, random\_state=42) |   The function randomly splits the data using the test\_size parameter. In this example, we now have a test set (test) that represents 33% of the original dataset. The remaining data (train) then makes up the training data. We also have the respective labels for both the train/test variables, i.e. train\_labels and test\_labels.  We can now move on to training our first model.  **Step 4 — Building and Evaluating the Model**  There are many models for machine learning, and each model has its own strengths and weaknesses. In this tutorial, we will focus on a simple algorithm that usually performs well in binary classification tasks, namely Naive Bayes (NB).  First, import the GaussianNB module. Then initialize the model with the GaussianNB() function, then train the model by fitting it to the data using gnb.fit():   |  | | --- | | ...  from sklearn.naive\_bayes import GaussianNB  # Initialize our classifier  gnb = GaussianNB()  # Train our classifier  model = gnb.fit(train, train\_labels) |   After we train the model, we can then use the trained model to make predictions on our test set, which we do using the predict() function. The predict() function returns an array of predictions for each data instance in the test set. We can then print our predictions to get a sense of what the model determined.  Use the predict() function with the test set and print the results:   |  | | --- | | ...  # Make predictions  preds = gnb.predict(test)  print(preds) |   Run the code and you’ll see the following results:  IMG_256  As you see in the Jupyter Notebook output, the predict() function returned an array of 0s and 1s which represent our predicted values for the tumor class (malignant vs. benign).  Now that we have our predictions, let’s evaluate how well our classifier is performing.  **Step 5 — Evaluating the Model’s Accuracy**  Using the array of true class labels, we can evaluate the accuracy of our model’s predicted values by comparing the two arrays (test\_labels vs. preds). We will use the sklearn function accuracy\_score() to determine the accuracy of our machine learning classifier.   |  | | --- | | ...  from sklearn.metrics import accuracy\_score  # Evaluate accuracy  print(accuracy\_score(test\_labels, preds)) |   we’ll see the following results:  IMG_256  As we see in the output, the NB classifier is 94.15% accurate. This means that 94.15 percent of the time the classifier is able to make the correct prediction as to whether or not the tumor is malignant or benign. These results suggest that our feature set of 30 attributes are good indicators of tumor class.  we have successfully built our first machine learning classifier. Let’s reorganize the code by placing all import statements at the top of the Notebook or script. The final version of the code should look like this:   |  | | --- | | from sklearn.datasets import load\_breast\_cancer  from sklearn.model\_selection import train\_test\_split  from sklearn.naive\_bayes import GaussianNB  from sklearn.metrics import accuracy\_score  # Load dataset  data = load\_breast\_cancer()  # Organize our data  label\_names = data['target\_names']  labels = data['target']  feature\_names = data['feature\_names']  features = data['data']  # Look at our data  print(label\_names)  print('Class label = ', labels[0])  print(feature\_names)  print(features[0])  # Split our data  train, test, train\_labels, test\_labels = train\_test\_split(features, labels, test\_size=0.33, random\_state=42)  # Initialize our classifier  gnb = GaussianNB()  # Train our classifier  model = gnb.fit(train, train\_labels)  # Make predictions  preds = gnb.predict(test)  print(preds)  # Evaluate accuracy  print(accuracy\_score(test\_labels, preds)) |   Now we can continue to work with our code to see if we can make our classifier perform even better. We could experiment with different subsets of features or even try completely different algorithms. | | |