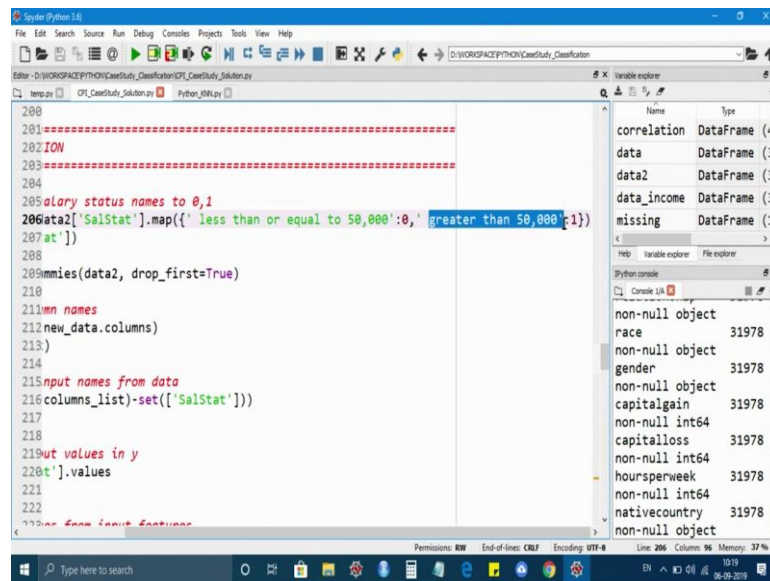


Python for Data Science
Prof. Raghunathan Rengasamy
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 24
Case Study - Classification

(Refer Slide Time: 00:16)



```
200
201=====
202TOW
203=====
204
205 salary status names to 0,1
206 data2['SalStat'].map({'less than or equal to 50,000':0, 'greater than 50,000':1})
207 at'})
208
209 mmies(data2, drop_first=True)
210
211 new names
212 new_data.columns
213 )
214
215 input names from data
216 columns_list=set(['SalStat'])
217
218
219 put values in y
220 t'].values
221
222
223 new from input features
```

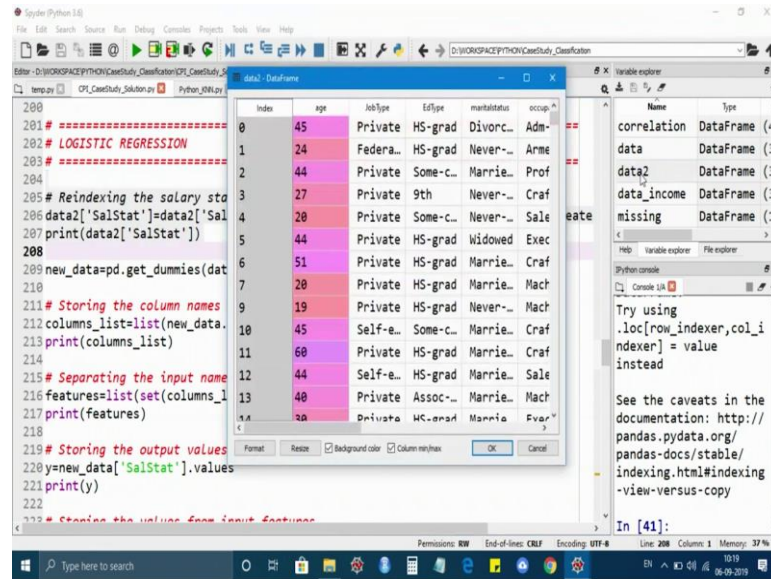
| Name | Type |
|-------------|-------------|
| correlation | DataFrame (|
| data | DataFrame (|
| data2 | DataFrame (|
| data_income | DataFrame (|
| missing | DataFrame (|

| Name | Type |
|-----------------|-------|
| non-null object | |
| race | 31978 |
| non-null object | |
| gender | 31978 |
| non-null object | |
| capitalgain | 31978 |
| non-null int64 | |
| capitalloss | 31978 |
| non-null int64 | |
| hoursperweek | 31978 |
| non-null int64 | |
| nativecountry | 31978 |
| non-null object | |

So, by exploring on the data we have looked at the distribution of each of the variables and the relationship that exists between the variables. So now, we are going to build a Logistic Regression Model, logistic regression is a machine learning classification algorithm that is used to predict the probability of a categorical dependent variable. So, using logistic regression we will build a classify model based on the available data. The first step that we are doing here is re indexing the salary status names to 0 and 1 because the machine learning algorithms cannot work with categorical data directly.

So, categorical data must be converted to numbers. So, we can assign 0 to less than or equal to 50000 and one to greater than 50000 using the map function and we are giving the values using the dictionary. Where we have mapped all the less than or equal to 50000 string values to 0 and greater than 50000 to an integer called 1, so let us run that.

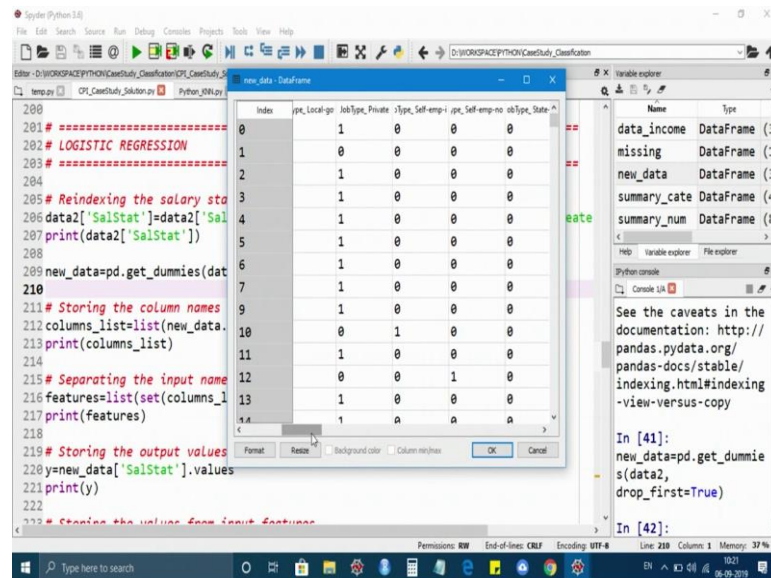
(Refer Slide Time: 01:23)



So when you open the data to data frame you will be able to see the salary status values as zeros and ones. So, the method that we used here is integer encoding. So, that we can invert the encoding later and get labels back from integer values such as in the case of making a prediction and using the pandas function to get_dummies. We can convert the categorical variables into dummy variables which is called as one hot encoding. It refers to splitting the column which has the categorical data to many columns depending on the number of categories present in the column.

So, let us see how to do that. So, we are using the get dummies function and inside the function we have specified the data frame name and I have stored the output to new data frame called new_data.

(Refer Slide Time: 02:16)



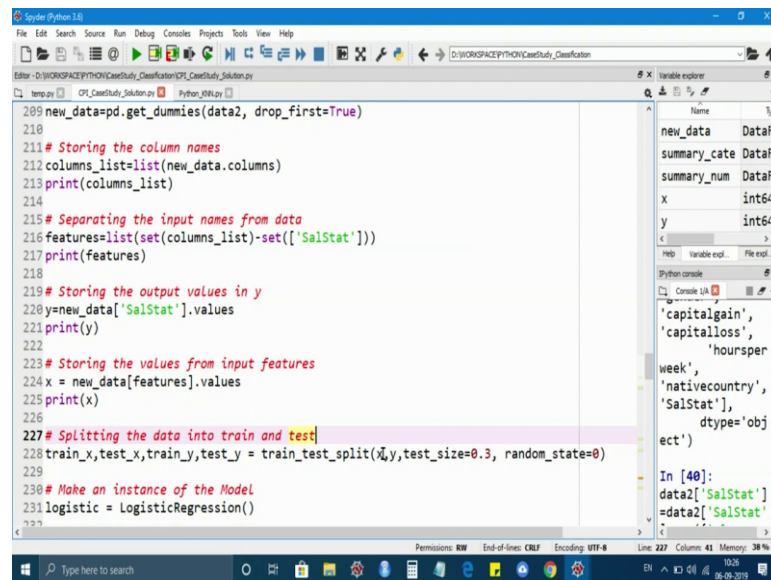
```
280
281 # =====
282 # LOGISTIC REGRESSION
283 # =====
284
285 # Reindexing the salary sta
286 data2['SalStat']=data2['Sal
287 print(data2['SalStat'])
288
289 new_data=pd.get_dummies(dat
290
291 # Storing the column names
292 columns_list=list(new_data.
293 print(columns_list)
294
295 # Separating the input name
296 features=list(set(columns_1
297 print(features)
298
299 # Storing the output values
300 y=new_data['SalStat'].values
301 print(y)
302
303 # Storing the values from input features
```

| index | job_Local-go | JobType_Private | Type_Self-emp-1 | job_Self-emp-no | jobType_State-1 |
|-------|--------------|-----------------|-----------------|-----------------|-----------------|
| 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 0 | 0 | 0 | 0 |
| 3 | 1 | 0 | 0 | 0 | 0 |
| 4 | 1 | 0 | 0 | 0 | 0 |
| 5 | 1 | 0 | 0 | 0 | 0 |
| 6 | 1 | 0 | 0 | 0 | 0 |
| 7 | 1 | 0 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 0 | 0 |
| 10 | 0 | 1 | 0 | 0 | 0 |
| 11 | 1 | 0 | 0 | 0 | 0 |
| 12 | 0 | 0 | 1 | 0 | 0 |
| 13 | 1 | 0 | 0 | 0 | 0 |
| 14 | 1 | 0 | 0 | 0 | 0 |

So, when we open the new_data if you see here if you check the data frame each column contains 0 or 1. So, let us take the example of the JobType variable, JobType earlier had seven categories to it, so each of the categories have been splintered in to columns. So, that the column has only the value 0 and 1. For example, the first row sees it has one under the column called JobType_private, then it means that the first row corresponds to the private JobType. Similarly you can interpret for the other values. So, now, we have mapped all the string values to integer values, so that we can work with any machine learning algorithms.

Next we are going to select the features where we need to divide the given columns into two types that is one having independent variables and one having the dependent variables .The dependent variable we are going to represent using letter called y and the independent variables being represented as x.

(Refer Slide Time: 03:22)



The screenshot shows a Jupyter Notebook with the following code in the editor:

```
209 new_data=pd.get_dummies(data2, drop_first=True)
210
211 # Storing the column names
212 columns_list=list(new_data.columns)
213 print(columns_list)
214
215 # Separating the input names from data
216 features=list(set(columns_list)-set(['SalStat']))
217 print(features)
218
219 # Storing the output values in y
220 y=new_data['SalStat'].values
221 print(y)
222
223 # Storing the values from input features
224 x = new_data[features].values
225 print(x)
226
227 # Splitting the data into train and test
228 train_x,test_x,train_y,test_y = train_test_split(x,y,test_size=0.3, random_state=0)
229
230 # Make an instance of the Model
231 logistic = LogisticRegression()
```

The Variable Explorer on the right shows the following variables:

| Name | Type |
|--------------|-----------|
| new_data | DataFrame |
| summary_cate | DataFrame |
| summary_num | DataFrame |
| x | int64 |
| y | int64 |

The Console output shows the following:

```
In [40]:
data2['SalStat']
= data2['SalStat']
```

For that we are going to store the column names as a list, by accessing the column names from the new_data dataframe and we are storing it under a list called columns_list. So, it will have only the column names from the new data dataframe. So, now, we need to separate the input variables from the data.

So, let us exclude the salary status variable from the columns_list and store it as features. So, from the list columns list we are going to exclude only the variable salary status. So, that we will have only the independent variables and storing that output to variable called features. So, let us print features and see what it has. So, it basically has all the column names from the columns list, where we do not have only the salary status value.

So, if you look at the variable explorer originally the columns_list had 95 values, when we excluded the salary status variable from it the features list *have only 95* the features list have only 94 values. But here we just have the column names now, so let us store the output values in y using the .values, the .values can be used to extract the values from data frame or a vector. So, I am going to extract the values from salary status and store it into y.

Similarly, we can do that to extract the values from features. So, that you will have all the corresponding values from the independent variables and then we can store it under the variable called x. Now x and y contains the integer values, where the x dimension is 30162 cross 94 and the y dimension is 30162. Next we are going to split the data into

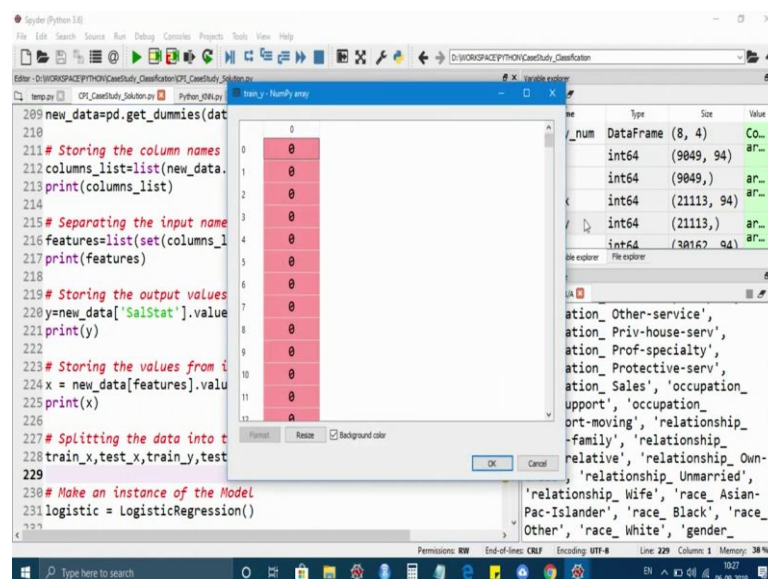
train and test we have to divide the data set into a training set and a test set. So, that we can build the model on train set and reserve some part of data to test the model on. This can be done by using the train_test_split command, the input parameters for the train test split command would be x and y, where x represents the input values and y represents the output values and then comes the test size.

I have given it as 0.3 that represents the proportion of the data set to include in the test split. The next is the random state I have given a value I have given an integer value as 0. So, here random state is the seed used by the random number generator, so that each and every time you run this line while sampling same set of samples will be chosen. If you have not set the random seed then different set of samples will be chosen for the analysis and I have saved the output into four different variables train_x test_x train_y and test_y.

So, let us see the dimension and what are the values that each of the variables have. So, first what we have done here is we have splitted our data into train and test, under train we have two sets of data which has only the input variables and the other will have the only the output variable.

Similarly, we have done it for the test set as well. So, the test will contain the input variable separately and the output variable separately. So, if you look at the train_x size 70 percentage of the data is contained in train_x which is 21113 observation with 94 variables.

(Refer Slide Time: 07:38)



The screenshot shows a Jupyter Notebook with the following code:

```
209 new_data=pd.get_dummies(data)
210
211 # Storing the column names
212 columns_list=list(new_data.columns)
213 print(columns_list)
214
215 # Separating the input name
216 features=list(set(columns_list)-set(y))
217 print(features)
218
219 # Storing the output values
220 y=new_data['SalePrice'].values
221 print(y)
222
223 # Storing the values from input
224 x = new_data[features].values
225 print(x)
226
227 # Splitting the data into train and test
228 train_x, test_x, train_y, test_y = train_test_split(x, y, test_size=0.3, random_state=0)
229
230 # Make an instance of the Model
231 logistic = LogisticRegression()
```

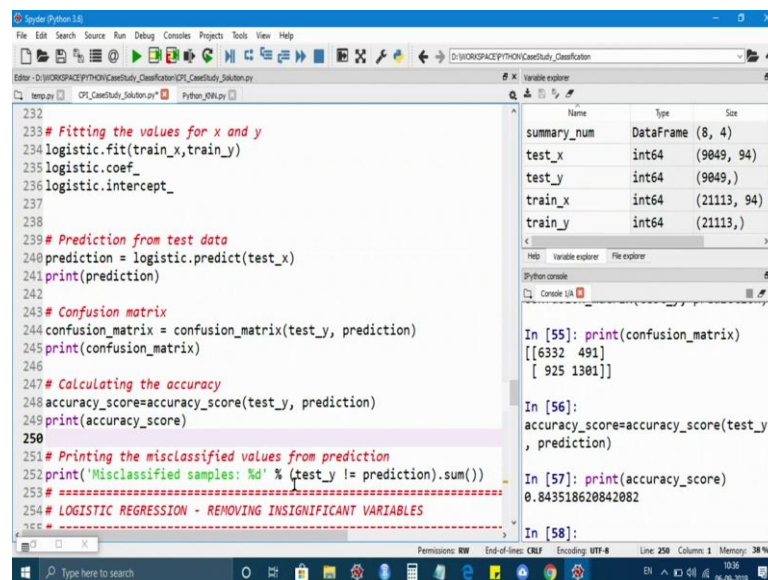
Two variable inspection windows are open:

- train_x - NumPy array**: Shows a 1D array of 0s.
- new_data**: A DataFrame with 8 rows and 4 columns. The columns are 'int64', 'int64', 'int64', and 'int64'. The values are (9849, 94), (9849, 94), (21113, 94), (21113, 94), and (38167, 94).

And the train_y will have only the output column which will have only the values as zeros and ones. Similarly if you look at the test_x size it has 30 percentage of the data to it that is 9049 observations with 94 variables and the test wise dimension is about 9400 9049 observation with only the salary status as output variable. Now you have split out the data into train set and test set.

So, next we are going to create a logistic regression classifier instance using the LogisticRegression function. So, here is the function that is logistic regression. So, let us make an instance of the logistic regression classifier.

(Refer Slide Time: 08:25)



The screenshot shows the Spyder Python IDE with a file named 'CPI_Classify_Solution.py'. The code in the editor includes comments and commands for fitting a logistic regression model, predicting on test data, calculating a confusion matrix, and determining accuracy. The variable explorer on the right shows the data types and sizes for 'summary_num', 'test_x', 'test_y', 'train_x', and 'train_y'. The Python console at the bottom displays the output of the code, including the confusion matrix and the accuracy score.

```
232
233 # Fitting the values for x and y
234 logistic.fit(train_x,train_y)
235 logistic.coef_
236 logistic.intercept_
237
238
239 # Prediction from test data
240 prediction = logistic.predict(test_x)
241 print(prediction)
242
243 # Confusion matrix
244 confusion_matrix = confusion_matrix(test_y, prediction)
245 print(confusion_matrix)
246
247 # Calculating the accuracy
248 accuracy_score=accuracy_score(test_y, prediction)
249 print(accuracy_score)
250
251 # Printing the misclassified values from prediction
252 print('Misclassified samples: %d' % (test_y != prediction).sum())
253 # =====
254 # LOGISTIC REGRESSION - REMOVING INSIGNIFICANT VARIABLES
255
```

| Name | Type | Size |
|-------------|-----------|-------------|
| summary_num | DataFrame | (8, 4) |
| test_x | int64 | (9049, 94) |
| test_y | int64 | (9049,) |
| train_x | int64 | (21113, 94) |
| train_y | int64 | (21113,) |

```
In [55]: print(confusion_matrix)
[[6332 491]
 [ 925 1381]]

In [56]:
accuracy_score=accuracy_score(test_y
, prediction)

In [57]: print(accuracy_score)
0.843518620842082

In [58]:
```

Then we can fit the model on the train set using the fit function and input for the fit function should be the input variables and the output variable. So, we have that under train_x and train_y respectively and I am using the LogisticRegression instance here to fit the model on train data.

So, now we have fitted the model on to the train data. So, you can extract some of the attributes from the logistic regression classify model by using the model name that is logistic and by using the .operator you will be able to access some of the attributes from it. So, let us get the coefficients of the logistic regression model. So, so in the console you will be able to get the coefficient values of all the variables. Similarly if you want to look at the intercept value you can also get that.

So, you can get that by accessing the intercept from the logistic regression model. So, the intercept is -3.8456. So, now we have built the model which will be used to classify the individual salary status as less than or equal to 50000 or greater than 50000. So, now we need to predict it on to a new data set, so that we can see how the model is performing. So, we have a data frame called `test_x` which the model has not seen yet. So, I am going to predict the model on to the `test_x` data frame using the `predict` function and I am saving my output to a variable called `prediction`. So, that my so that the predictions will be stored in to the variable called `prediction`.

So, let us print the prediction and see what it has ideally it should be having only the values are zeros and one, which basically gives you the salary status of the test data frame. So, now we have built a logistic regression model and we have also tested it on to a new data frame called `test_x` and we also got the predictions.

So now, we have to evaluate the model using the confusion matrix, confusion matrix is a table that is used to evaluate the performance of a classification model. The confusion matrix output gives you the number of correct predictions and the number of incorrect prediction and it will sum up all the values class wise. So, let us see how to use the confusion matrix in order to evaluate the model.

So, confusion matrix is the function the input should be just the actual and the predicted values, the actual salary status is under the vector `test_y` and the predictions are under the variable `prediction`. So, those can be the input for the confusion matrix and we are storing the output into object called confusion matrix. So, let us print the confusion matrix and see the output.

So, in the output you have four values, the diagonal values gives you the total number of correctly classified samples and the off diagonal values gives you the total number of wrongly classified samples. So, in the column what you have here is the predictions and in the row wise what you have is the what you have is the class for the actual and in the rows you have the actual classes.

For example, if the actual class is less than or equal to 50 and the model has predicted 6332 observations as less than or equal to 50000 but being less than or equal to 50000 is the actual class the model has predicted 491 observation as greater than 50000. Similarly given the salary status is greater than given the actual salary status is greater than 50000,

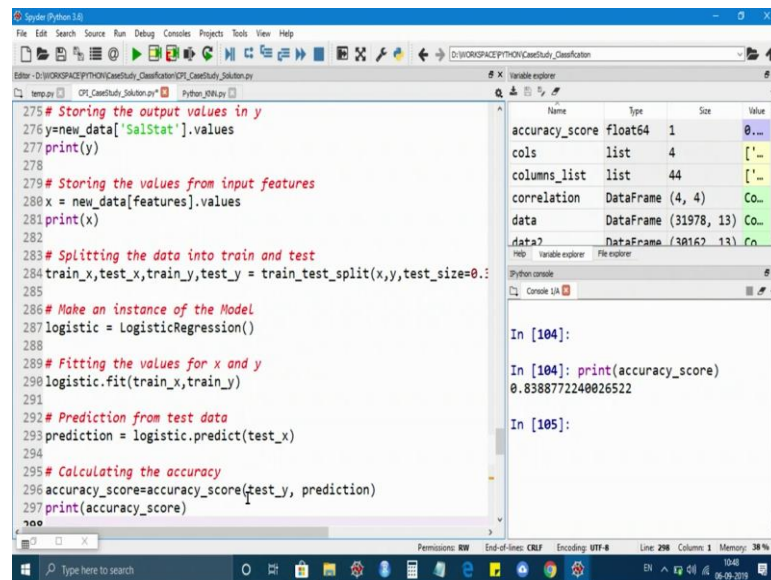
the model has predicted thousand 301 observation as greater than 50925 observations as less than or equal to 50000.

So, there are many misclassification secure the model has not classified all the observations correctly. So, using a measure called accuracy we will be able to get the accuracy score of the model that variable. So, it gives as an idea about how accurate our model is. So, you can get the accuracy for the model that we have built using the function called `accuracy_score`, the input can be the actual class and the predicted class and we are storing it into a variable called `accuracy_score` and when we print the accuracy, so the values turned out to be 0.8435. So, it means that 84 percentage of the time the model is able to classify the records correctly. So, now we have got an idea about how many misclassifications are there and what is the accuracy for the model that we have built.

We can also get the misclassified values from the predictions, like how many number of misclassifications are there you can do that by setting a simple condition you can get that by giving a simple condition. So, if you print the output it says 1416 observation has been misclassified. So, now you got an idea about how many misclassified number of samples were also there. So, now we have built a logistic regression model, but you can also improve the accuracy of the model by reducing the number of misclassified samples.

So, one of the method is by removing the insignificant variables, when we exclude more on the data we found out that there were sum in significant variables that might not contribute more in classifying the individual salary status. So, we are going to ignore those variables when we model it. So, what we are going to do here is we are going to build a new logistic regression model by removing all the insignificant variables.

(Refer Slide Time: 15:02)



The screenshot displays a Jupyter Notebook window with the following components:

- Code Editor:** Contains Python code for a logistic regression model. The code includes comments and operations for storing output values, splitting data into training and testing sets, fitting the model, and calculating accuracy. The code is as follows:

```
275 # Storing the output values in y
276 y=new_data['SalStat'].values
277 print(y)
278
279 # Storing the values from input features
280 x = new_data[features].values
281 print(x)
282
283 # Splitting the data into train and test
284 train_x, test_x, train_y, test_y = train_test_split(x, y, test_size=0.1)
285
286 # Make an instance of the Model
287 logistic = LogisticRegression()
288
289 # Fitting the values for x and y
290 logistic.fit(train_x, train_y)
291
292 # Prediction from test data
293 prediction = logistic.predict(test_x)
294
295 # Calculating the accuracy
296 accuracy_score=accuracy_score(test_y, prediction)
297 print(accuracy_score)
```
- Variable Explorer:** A table showing the types and sizes of variables in the current environment.

| Name | Type | Size | Value |
|----------------|-----------------------|------|--------------------|
| accuracy_score | float64 | 1 | 0.8388772248026522 |
| cols | list | 4 | ['...'] |
| columns_list | list | 44 | ['...'] |
| correlation | DataFrame (4, 4) | | Co... |
| data | DataFrame (31978, 13) | | Co... |
| data2 | DataFrame (30162, 13) | | Co... |
- Python Console:** Shows the execution of the code cells, including the output of the accuracy score calculation:

```
In [104]:
In [104]: print(accuracy_score)
0.8388772248026522
In [105]:
```

So, let us first map all the less than or equal to 50 to 0 and greater than to 1 and then store it to the original data frame salary status and then store it to the variable called salary status of the data frame data2 and then these are the variables that were insignificant that is gender, nativecountry, race and JobType and I am creating the list of columns and I am storing it under the list called cols and I am going to drop all these listed columns from the data frame data2, so that I am not going to consider all these variables when I model it.

And the new data frame is created by dropping out all the insignificant variables to proceed with the further analysis and after removing the columns we need to get the dummies of all the categorical variables. So, we are also doing that and the other steps remind the same like I have stored all the columns names in two columns_list and then I have separated the input names from the data by excluding only the salary status variable. So, features will have only the set of column names of the input variables and we are going to extract the values in y and we are going to extract the values from salary status and store it into y.

Similarly we can also extract the values from the features and store it as x and then again we are splitting the data into train and test. So, that we can build the model on train and we can use the test data frame to test the model on and then we are creating an instance

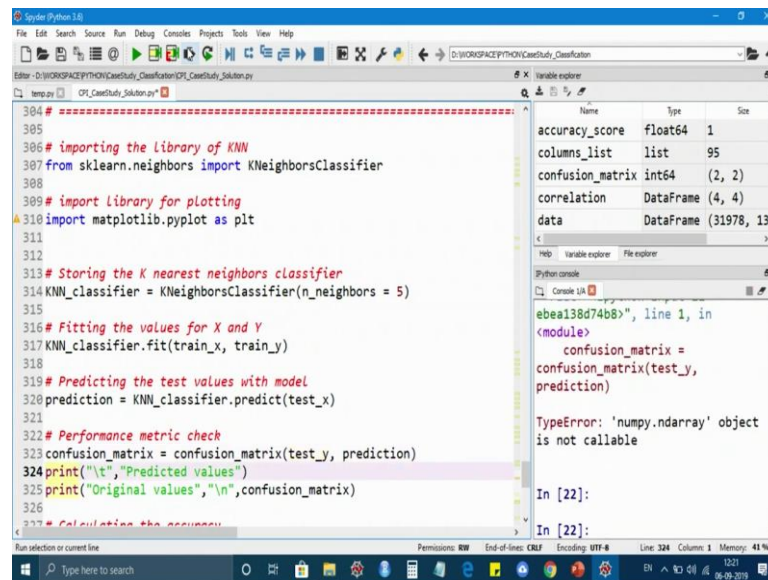
call `logistic` from the function `LogisticRegression`, which is used to build the logistic regression classifier model and we are using the instance created that is `logistic`.

We are going to fit the logistic regression model on the train data and after building the model we can predict the model on to the new data frame that is `test_x`. So, once we have got the prediction we are interested in getting or we are interested in improving the accuracy by removing all the insignificant variables. So, let us check whether the accuracy has improved by removing all the insignificant variables, we are going to get the accuracy using the accuracy score function.

So, when we print the accuracy score the accuracy is turned out to be 8.8388 which has higher than the accuracy score that we have got from the model where we have used all the variables. So, the accuracy has dropped down from 85 percent to 83 percent, because we have removed all the insignificant variables.

So, by removing the insignificant variables we are not getting a better model which gives us the better accuracy, when we remove any variables of course we are losing some information from the data. So, that could be the reason that is why we are getting a decrease in accuracy, but as compared to the model with all the variables it is not very low it is just a slight decrease in the accuracy. So, if you want to reduce the data collection part, if you want to retain only the significant variables which is used to classify the data. Then we can remove all the insignificant variables and keep this model as the final model.

(Refer Slide Time: 18:32)



The screenshot displays a Jupyter Notebook window with a code editor on the left and a variable explorer and console on the right. The code in the editor includes imports for KNeighborsClassifier and matplotlib, creating a KNN classifier with 5 neighbors, fitting it to training data, making predictions on test data, and printing the confusion matrix. The variable explorer on the right shows the following variables:

| Name | Type | Size |
|------------------|-----------|-------------|
| accuracy_score | float64 | 1 |
| columns_list | list | 95 |
| confusion_matrix | int64 | (2, 2) |
| correlation | DataFrame | (4, 4) |
| data | DataFrame | (31978, 13) |

The console on the right shows the execution of the code, including a warning about a deprecated function and a final error message: 'TypeError: 'numpy.ndarray' object is not callable'.

Now we have looked at the performance of the logistic regression model, so similarly we are going to build a KNN model, KNN classifier model to classify the records into any one of the categories of the salary status. So, to build a KNN model we are going to import k nearest neighbor classifier package from the sklearn neighbors. So, we have to import that, so after importing KNeighborClassifier we are importing the matplotlib library we are importing pyplot from matplotlib library as plt.

We will be using the matplotlib library to visualize some of the outputs of the k nearest neighbor classifier model so let us import that. So, after importing the necessary libraries we are creating an instance for k nearest neighbors, we are creating an instance for k nearest neighbor classifier using the function k nearest neighbors classifier and inside the function I have specified n neighbors as 5 that is the k value.

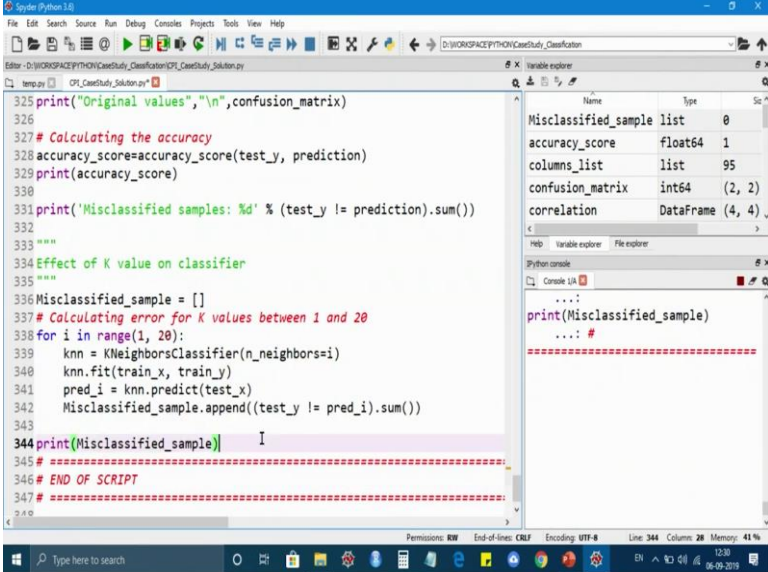
So, that it consider five neighbors when classifying the data into less than or equal to 50000 or greater than 50000. So, if it considers 5 neighbors, then it will take the majority classes from the 5 neighbors and then it will classify the new data based on the majority voting method and we are also saving the output to an object called KNN classifier. So, KNN classifier is the model, so using the instance that we have created using the k neighbor's classifier that is KNN classifier. We are going to fit the model on to a data frame called on to the train data frame using the fit function and input to the fit function should be the input values and the output values of from the train data frame.

So, now we have build the classify model using the KNN algorithm now let us see how the model is performing in a new data frame. So, we have a test data frame so that we can test a model on. So, I am going to predict the model on to a new data frame called test using the predict function.

So, the input should be the input values of the test data frame and I am saving the output into a variable call prediction. So, the prediction will have the values of the salary status as zeros and one, where 0 represents less than or equal to 50 and 1 represent greater than or equal to 50000. So, now, we have predicted the values and you have also seen the values from the prediction output. Now, it is time to check how good of a model is performing using the confusion matrix.

So, the input can be the actual values and the predicted values of the salary status, which is there under test_y and prediction respectively and we have also save that to a variable called confusion_matrix.

(Refer Slide Time: 21:25)



```
325 print("Original values", "\n", confusion_matrix)
326
327 # Calculating the accuracy
328 accuracy_score=accuracy_score(test_y, prediction)
329 print(accuracy_score)
330
331 print('Misclassified samples: %d' % (test_y != prediction).sum())
332
333 """
334 Effect of K value on classifier
335 """
336 Misclassified_sample = []
337 # Calculating error for K values between 1 and 20
338 for i in range(1, 20):
339     knn = KNeighborsClassifier(n_neighbors=i)
340     knn.fit(train_x, train_y)
341     pred_i = knn.predict(test_x)
342     Misclassified_sample.append((test_y != pred_i).sum())
343
344 print(Misclassified_sample)
345 # =====
346 # END OF SCRIPT
347 # =====
```

The screenshot shows a Python IDE with a code editor on the left and a variable explorer on the right. The code editor contains Python code for KNN classification and evaluation. The variable explorer shows the following variables:

| Name | Type | Size |
|----------------------|-----------|--------|
| Misclassified_sample | list | 0 |
| accuracy_score | float64 | 1 |
| columns_list | list | 95 |
| confusion_matrix | int64 | (2, 2) |
| correlation | DataFrame | (4, 4) |

The Python console on the right shows the output of the code, including the print statement for Misclassified_sample.

So, now, let us look at the output of the confusion matrix. So 6338 observations have been classified properly as less than or equal to 51285 observations are correctly classified as greater than or equal to 50000 with the actual class. When the actual salary status is greater than 50, but the model has classified it as less than or equal to 50. So, now we have using the confusion matrix we have got an idea about how many

misclassified samples are there and how many correctly classified and how many correct predictions are there.

We can also look at the accuracy score using the accuracy score function we can also look at the accuracy of the model that we have built using the accuracy score function and the input to the accuracy score function should be the actual and the predicted classes of the salary status. So, let us print the accuracy score.

So, the accuracy of the KNN model is 0.8424, 84 percentage of the time the KNN model is able to classify the records correctly. If you want to check how many number of misclassified samples are there you can also check that using the same condition that you have already used for the logistic regression, when we check the output the misclassified samples are 1456 for the KNN model that we have built.

So, now under the neighbors argument while we are building the KNN model we have just randomly fixed a k value as 5. So, here we are trying to calculate the error for k values between 1 and 20 by iterating through for loop. So, I have given the range as 1 to 20, so that the k neighbors classifier will takes the k value from 1 to 20 and builds the model based on the k value, since I have given i under the neighbors.

So, once you have created an instance called KNN we are going to fit the model on the train data frame using the fit function and the train_x becomes the input values and the train_y as the output value. So, once we have build the model you are going to predict it on to a new data frame called test test_x using the predict function. So, all the creation of instance building the model and predicting it on test are happening inside the for loop.

So, that for each and every value of k you will get a different models and once we have predicted and the predictions are stored into a variable called pred_i. So, that you will have the predictions under pred_i and in the next step we are going to look at the number of misclassified samples when we fix the k values from 1 to 20. So, you will have the misclassified samples for each and every value of k that we have fixed here.

So, let us run the whole for loop and see what is the output. So, the print so the misclassified samples will give you the number of misclassified sample with respect to each values of k. So, based on the output you will be able to arrive at a k value for your k

nearest neighbor model, wherever you have the less number of misclassified samples you can fix that as the k value to build any k to build the KNN model.

(Refer Slide Time: 24:53)

```
326
327 # Calculating the accuracy
328 accuracy_score=accuracy_score(test_y, prediction)
329 print(accuracy_score)
330
331 print('Misclassified samples: %d' % (test_y != prediction).sum())
332
333 """
334 Effect of K value on classifier
335 """
336 Misclassified_sample = []
337 # Calculating error for K values between 1 and 20
338 for i in range(1, 20):
339     knn = KNeighborsClassifier(n_neighbors=i)
340     knn.fit(train_x, train_y)
341     pred_i = knn.predict(test_x)
342     Misclassified_sample.append((test_y != pred_i).sum())
343
344 print(Misclassified_sample)
345 # =====
346 # END OF SCRIPT
347 # =====
348
```

| Name | Type | Size |
|----------------------|-----------|--------|
| Misclassified_sample | list | 19 |
| accuracy_score | float64 | 1 |
| columns_list | list | 95 |
| confusion_matrix | int64 | (2, 2) |
| correlation | DataFrame | (4, 4) |

```
...:
...:
...:
print(Misclassified_sample)
...: #
=====
[1763, 1509, 1510, 1431, 1456,
1445, 1461, 1450, 1459, 1446,
1452, 1423, 1440, 1413, 1440,
1401, 1418, 1418, 1442]
In [36]:
```

So, from the output it is clear that for the k value 16 we are getting the number of misclassified samples as 1401. So, if we fix the k value of 16 then the model was performing good, because we are getting a less number of misclassification. Then we can fix this as a k value when we build a when we want to improve the model further. So, that so that we will not end up in misclassifying the salary status.

So, in this case study we have looked at two algorithms, one is logistic regression and the other one is KNN both of the algorithms gives you the same performance when we looked at the accuracy and in terms of number of misclassification.

But on the whole the logistic regression is performing better with the accuracy of 85 percent that when the KNN model gives you the accuracy of 84 percent but we can also improve the model further by dropping few more insignificant variables in the logistic regression as well as in the KNN classifier model because in the KNN classifier model that we have built here is only with is only including all the variables; we have not excluded this insignificant variable and check the accuracy. So, we can also do that to improve the performance of the classifier model that we have built.