

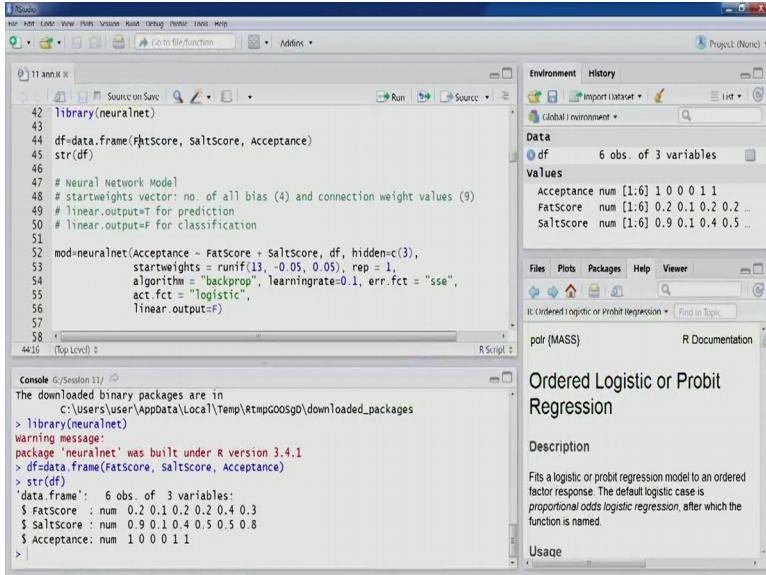
**Business Analytics & Data Mining Modeling Using R**  
**Dr. Gaurav Dixit**  
**Department of Management Studies**  
**Indian Institute of Technology, Roorkee**

**Lecture - 56**  
**Artificial Neural Network-Part IV**

Welcome to the course Business Analytics and Data Mining Modeling Using R. So, in previous we lecture lectures we have been discussing neural artificial neural networks and in particular in previous lecture we started our modeling exercise using this small data set related to cheese samples and there is you know the acceptance or rejection based on two predictors fat score and salt score. So, we were at this point, we were trying to build this model. So, let us start again.

So, data frame we had already created in the previous lecture. So, the same date frame you can see we are going to use the data frame 6 observations 3 variables.

(Refer Slide Time: 00:51)

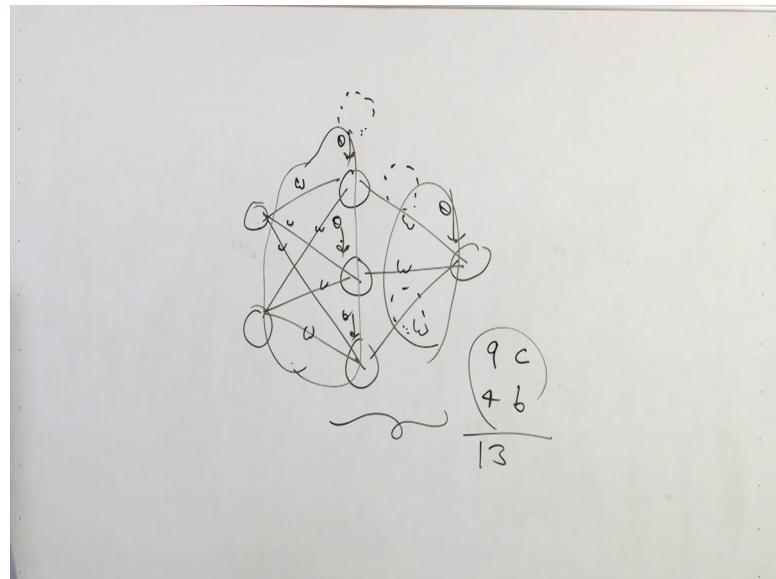


The screenshot shows the RStudio interface with the following details:

- Code Editor:** Shows R script code for building a neural network model. The code includes loading the 'neuralnet' library, creating a data frame 'df' from 'Fatscore' and 'Saltscore' columns, and defining a neural network model 'mod' with one hidden layer of 3 neurons, using backpropagation as the algorithm, and logistic activation function.
- Environment View:** Displays the data frame 'df' with 6 observations and 3 variables: Acceptance, Fatscore, and Saltscore.
- Console View:** Shows the R session output, including the warning message that 'neuralnet' was built under R version 3.4.1.
- Help View:** Shows the help documentation for 'pair(MASS)' and 'Ordered Logistic or Probit Regression'.

The particular neural network architecture that we are going to use is the same that we had used in previous lectures this is the so the architecture is this one.

(Refer Slide Time: 01:12)



So, this was the architecture that we had used in previous lectures for this particular example, right. So, these are the connections, and then further then bias values. So, we can see these connections and bias values, these this is the architecture that we are going to use. So, as I talk about we can certainly do certain experimentation with this particular architecture, we can certainly add more hidden more hidden layers'.

So, we can have one more hidden layer here with two nodes. So, this kind of experimentation we can always perform. However, what has been seen that typically one hidden layer is sufficient to model even the most complicated complex relationships. So, we typically start with one hidden layer and typically the performance is higher for higher for the one hidden layer networks.

Again in terms of how many nodes that we should be using in a particular hidden layer that is also of course, we can experiment with that part also. So, we can of course, have you know one more node and that experimentation and the performance of that particular network we can always compare it with. So, we can always build different candidate network model and we can always check which one is performing better, and those kind of experimentation with the network architecture can also be performed. However, for our this illustration for this exercise we are sticking to this particular network 2 nodes, 3 nodes and 1 node in the input layer hidden, hidden layer and output layer respectively.

So, let us discuss further about this so the package as we talked about is the neural net and the function is neural net that we are going to use to build our neural network model. So, the first argument is the formula for the model equation as you can see, acceptance is the output variable outcome variable then we have two predictors fat score and salt score. And then data is coming from the data frame df and you can see the next argument is hidden, so that is a number of hidden layers and you know that is the number of nodes in different hidden layers. So, we have just 1 hidden layer and we have 3 nodes, we have just mentioned 3 here. However, if you have we have more number of hidden layers and different number of nodes there, so that can we specified using this particular vector.

Then we have a start weights. So, this is for the initialization. So, like we did in our previous exercise be the same data set so we can see we need 13 values. So, this we can understand we have number of bias values 4. So, we have 4 bias values and here for every node we have it is connected with the nodes of next layer. So, 2 you know 3 into 2, 6 plus 3, so we have 9 connections and we have 4 bias values. So, in total we need 13 initialization; 13 values we need to initialize.

So, the same thing is mentioned here in the start weights. You can see start weights run if 13 values, and as we talked about that typically we initialize these values from minus 0.5 to 0.5. So, the same thing is mentioned here. So, this will be used this these particular values are going to be used for the initialization step. And then when we see another argument rep that is for number of repetition a number of repetition that we want to number of times that we want to train our model, right.

So, we just want to run it you know once right. So, this we if you understand finding more detail about this particular function you can go here in the help section neural net and you will get more details about this particular function.

(Refer Slide Time: 05:31)

The screenshot shows the RStudio interface with the following details:

- Code Editor:** Displays the R script for training a neural network. It includes the `library(neuralnet)` command, creation of a data frame `df` from variables `FatScore`, `SaltScore`, and `Acceptance`, and the call to `mod<-neuralnet(Acceptance ~ FatScore + SaltScore, df, hidden=c(3), startweights = runif(13, -0.05, 0.05), rep = 1, algorithm = "backprop", learningrate=0.1, err.fct = "sse", act.fct = "logistic", linear.output=F)`. The script also includes a `str(df)` command.
- Console:** Shows the output of the R session, including the loading of the `neuralnet` package, the creation of the data frame, and the resulting structure of `df` (6 obs. of 3 variables).
- Help Documentation:** The right pane shows the `?neuralnet` help page. It describes the function as used for training neural networks using backpropagation, resilient backpropagation (RPROP), or without weight backtracking (Riedmiller and Braun, 1993) or the modified globally convergent version (GRPROP) by Anastasiadis et al. (2005). It mentions flexible settings through custom-choice of error and activation function, and the calculation of generalized weights (Intrator O. and Intrator N., 1993).

You can see all these arguments and for example, specifically we were discussing a rep.

(Refer Slide Time: 05:34)

The screenshot shows the RStudio interface with the following details:

- Code Editor:** Displays the same R script as the previous screenshot, including the `library(neuralnet)` command, data frame creation, and the `mod<-neuralnet` call.
- Console:** Shows the same output as the previous screenshot, including the package loading and data structure.
- Help Documentation:** The right pane shows the `?neuralnet` help page, which is identical to the one in the first screenshot. It describes the function's purpose, available algorithms, and specific implementation details.

So, you can see number of repetitions for the neural networks training.

(Refer Slide Time: 05:35)

The screenshot shows the RStudio interface with the following details:

- Code Editor:** Shows the R script "11.R" with the following code:

```
library(neuralnet)
df<-data.frame(FatScore, SaltScore, Acceptance)
str(df)
# Neural Network Model
# startweights vector: no. of all bias (4) and connection weight values (9)
# linear.output=F for prediction
# linear.output=F for classification
mod<-neuralnet(Acceptance ~ FatScore + SaltScore, df, hidden=c(3),
                 startweights = runif(13, -0.05, 0.05), rep = 1,
                 algorithm = "backprop", learningrate=0.1, err.fct = "sse",
                 act.fct = "logistic",
                 linear.output=F)

```

- Environment View:** Displays the global environment with the variable "df".

values
Acceptance num [1:6] 1 0 0 0 1 1 FatScore num [1:6] 0.2 0.1 0.2 0.2 ... SaltScore num [1:6] 0.9 0.1 0.4 0.5 ...
- Console View:** Shows the command history and output:

```
library(neuralnet)
df<-data.frame(FatScore, SaltScore, Acceptance)
str(df)
# Neural Network Model
# startweights vector: no. of all bias (4) and connection weight values (9)
# linear.output=F for prediction
# linear.output=F for classification
mod<-neuralnet(Acceptance ~ FatScore + SaltScore, df, hidden=c(3),
                 startweights = runif(13, -0.05, 0.05), rep = 1,
                 algorithm = "backprop", learningrate=0.1, err.fct = "sse",
                 act.fct = "logistic",
                 linear.output=F)
```

The console also displays a warning message about the 'neuralnet' package being built under R version 3.4.1.
- Help View:** Shows the help documentation for the 'neuralnet' function.

So, in this case we just want one. If you specify more than one then of course, you will have in a way you will have two candidate models. So, that other actually based on two runs. So, an every run the results might be might change slightly as we have been talking about that machine learning algorithm or data driven models they are sensitive to data. So, therefore, you know every run results might change. So, from those runs we can always let the best model.

However, in this particular case for the illustration we are just running it building our model just once.

(Refer Slide Time: 06:29)

The screenshot shows the RStudio interface with the help documentation for the `neuralnet` function open. The code in the script pane is as follows:

```
library(neuralnet)
df<-data.frame(FatScore, SaltScore, Acceptance)
str(df)
# Neural Network Model
# startweights vector: no. of all bias (4) and connection weight values (9)
# linear.output=T for prediction
# linear.output=F for classification
mod<-neuralnet(Acceptance ~ FatScore + SaltScore, df, hidden=c(3),
                 startweights = runif(13, -0.05, 0.05), rep = 1,
                 algorithm = "backprop", learningrate=0.1, err.fct = "sse",
                 act.fct = "logistic",
                 linear.output=F)
str(mod)
```

The help pane on the right shows the `algorithm` argument with its description: "a string containing the algorithm type to calculate the neural network. The following types are possible: 'backprop', 'rprop+', 'rprop-', 'sag' or 'slr'. 'backprop' refers to backpropagation, 'rprop+' and 'rprop-' refer to the resilient backpropagation with and without weight backtracking, while 'sag' and 'slr' induce the usage of the modified globally convergent algorithm".

Now the algorithm that we can select, so there are multiple options here, so that we can you can see in the help section. So, this particular argument algorithm is here.

(Refer Slide Time: 06:36)

The screenshot shows the RStudio interface with the help documentation for the `neuralnet` function open. The code in the script pane is identical to the previous screenshot:

```
library(neuralnet)
df<-data.frame(FatScore, SaltScore, Acceptance)
str(df)
# Neural Network Model
# startweights vector: no. of all bias (4) and connection weight values (9)
# linear.output=T for prediction
# linear.output=F for classification
mod<-neuralnet(Acceptance ~ FatScore + SaltScore, df, hidden=c(3),
                 startweights = runif(13, -0.05, 0.05), rep = 1,
                 algorithm = "backprop", learningrate=0.1, err.fct = "sse",
                 act.fct = "logistic",
                 linear.output=F)
str(mod)
```

The help pane on the right shows the `algorithm` argument with its description: "a string containing the algorithm type to calculate the neural network. The following types are possible: 'backprop', 'rprop+', 'rprop-', 'sag' or 'slr'. 'backprop' refers to backpropagation, 'rprop+' and 'rprop-' refer to the resilient backpropagation with and without weight backtracking, while 'sag' and 'slr' induce the usage of the modified globally convergent algorithm".

You can see many options here. So, we have back prop that is traditional back propagation algorithm. Then we have R prop plus, R prop minus and other you know variance. So, all these could be used to build our neural network model; however, for our exercise we are using back prop. Then certain arguments in this particular function are depend on the algorithm that is being used.

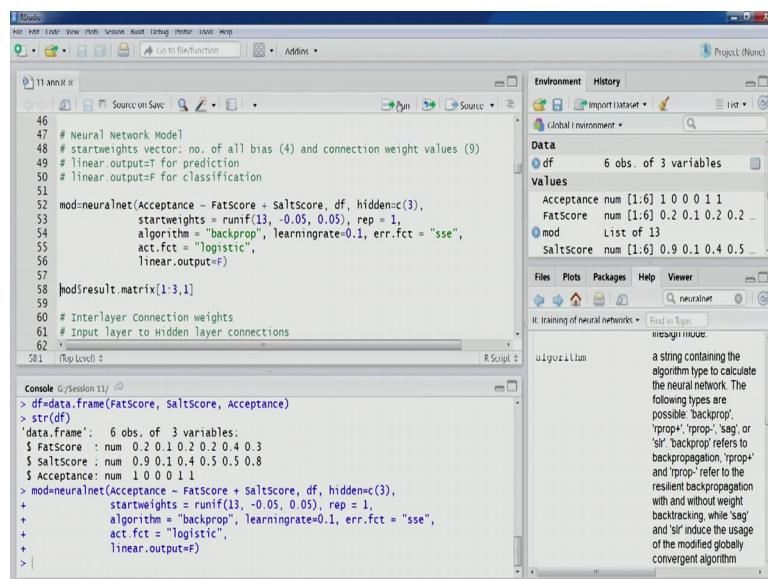
So, for example, in this case back prop we might specify one more argument that is learning rate, right. So, we discussed about the learning rate that the value the constant value, that could be used for the you know to control the amount of learning that happens and you know that happens from previous iteration or in each iteration. So, this is 0.1, so right.

So, the whatever formula that we saw the updation formulas that we saw in the previous lecture for thetas and weights ah, so there it was the values were or the addition was the learning rate into the error value. So, you can see 10 percent of that particular value is being used to learn.

Now, we have error dot fct; so this function again can be used. So, this is sse, so this can be used to check the overall model error then we have act dot fct; so this is activation function which is nothing, but what we have been calling as transfer function. So, this activation function is actually the transfer function. So, as we talk about different alternatives the logistic is the most popular. So, that is being used here.

Then as we talked about in previous lecture linear output is the argument that is to be specified as false if we are building a classification model and if you are building a application model then this has to be specified as true. So, let us run this code. Let us run this and we will have our model.

(Refer Slide Time: 08:35)



The screenshot shows the RStudio interface with the following details:

- Code Editor:** Displays R code for training a neural network. The code includes:
  - Reading data from a CSV file: `> df<-read.csv("C:/Session 11/fatScore_SaltScore_Acceptance.csv")`
  - Viewing the data structure: `> str(df)`
  - Training the neural network: `> mod<-neuralnet(Acceptance ~ FatScore + SaltScore, df, hidden=c(3), startweights = runif(13, -0.05, 0.05), rep = 1, algorithm = "backprop", learningrate=0.1, err.fct = "sse", act.fct = "logistic", linear.output=F)`
- Console:** Shows the execution of the R code and the resulting output.
- Environment View:** Shows the global environment with variables like `df`, `mod`, and `values`.
- Help View:** Provides detailed information about the `algorithm` argument in the `neuralnet` function.

Now in the model we many values are written. So, one of them is result dot matrix that will actually have the values of you know bias values and connection weights and few more a few more parameters about the model.

So, you can see here we are just looking at first 3 values, so first 3 values of this particular matrix. So, first column so, you can see. So, first 3 values are actually about error. So, this error is actually based on. So, this is actually sse value because we had used sse here; so we have other options also for model error. So, as you can see here sse and ce in the help section and you can see ce is cross entropy error and as a some of square error. So, these are the option that could be used.

Then we have reached this threshold. So, that is 0.0098; so that is about 0.01. So, threshold we had not specified, however the default value for the threshold is there. So, that we can see you can see threshold is 0.01. So, the threshold has these two this level therefore the training process was it stopped, right. So, the threshold is again based on this the error value that we talked about and the steps that have been taken 6 steps were required. However, if as we talked about if the model is not able to converge then probably the training process would be stopped by the limit that is specify the number of steps.

However, by default this limit is quite high; so therefore, there is a you know good chance that model will converge you can see the default value is step max in the help section, you can see steps max it is 1 e plus 0 5. So, that is 10000, I think that is 10000 steps more than so that is more than 10000 steps 1 lakh, 1 lakh steps. So, that could be used. So, those number of steps put the limit. So, in this particular case only 6 steps have been used.

Now, let us move forward. So, next thing that we would like to understand is the interlayer connection weights, so which is nothing but the information about these values. So, these are these are some of the weights so we would like to see from the model; the final model that we have what are these values thetas and weights.

So, first we will look at these values and these weights and theta combination and then here these value. So, typically because we have one hidden layer if we had a more hidden layer and again and we would we can; so for every inter layer combination, so

input layer 2 first hidden layer in this case we have just one hidden layer. So, from input layer to this hidden layer we will have few weights and bias values.

Now next is from hidden layer to output layer; so again we have some weights and bias value. So if we had more hidden layer, so we would have more such combinations. So, inter layer connections weights and bias values we want to have a look. So, first we look at the input layer to hidden layer connection. So, this is the matrix that we can this is actually list that is returned when the model is built. So mod\$weights. So, within this the you can see in the list the first element of this list is actually nothing, but a matrix for storing these weights right.

So, by default these values are stored using the default row number and column number 1 2 3. However, I have changed the dimension names that is row names and column names for this particular matrix. So, that we are able to understand which particular value is for which particular connection or by aspect or you know for which bias value for which particular node.

So, we this is the code; so you can see I am using dim names function and we use this function allows us to change the row names and column names of a particular matrix or data frame. So, in this case we can see the list is to be supplied. So list first early first element is always the row names and then the second element is the for the column names.

(Refer Slide Time: 13:25)

The screenshot shows the RStudio interface with the following details:

- Code Editor:** Shows the R script being run. It includes code for setting up a logistic activation function, defining the input and hidden layers, and training the neural network. The script uses the `mod` object to store the results.
- Console:** Displays the output of the R code. It shows the execution of the neural network training command, which prints various parameters like learning rate, steps, and weights.
- Environment:** Shows the variables available in the global environment, including `df`, `mod`, and `mod$weights`.
- Plots:** Shows a small neural network diagram.
- Packages:** Shows the loaded packages: `neuralnet`.
- Help:** Shows the help documentation for the `neuralnet` function.
- Viewer:** Shows the results of the neural network training.

So, let us execute this code and you can see this is the result; you can see bias values for node 3 is this one, node 4 is this one; so you can see for node 5, similarly a node 1, which is also corresponding to the predictor fat. So, the connection weights are specified so from node 1 to node 3 and node 4 and node 5 these are the weights. Then node 2 that is corresponding to the predicted salt and the connections to node 3, node 4 and node 5 and the connection weights we can see here.

So, a specific connection weights and their values, the bias values and connection weights values we can access in this fashion. Similarly from hidden layer to output layer also we have 3 weights and 1 bias value, so 4 values. So, that also we can access and in the similar fashion. So, again I am changing the dimension in here again for this particular second element of weights a list. So, again as you can see bias the list is being supplied with first element being the row names and the second element being the you know column names. So, let us execute this code.

(Refer Slide Time: 14:40)

The screenshot shows the RStudio interface with several panes:

- Top Bar:** File, Edit, View, Plots, Session, Build, Debug, Profile, Tools, Help.
- Left pane:** A code editor with the file "11 ann.R" open, containing R code for training a neural network. The code includes matrix operations, dimension names, and training record classification.
- Right pane:** A "Project (None)" view with tabs for Environment, History, Global environment, Data, Values, and Files. It also shows the current R session's environment and a search bar.
- Bottom pane:** A "Console" window showing the output of the R code, including the neural network structure and training progress.

Now you would see that the row names bias, now you see the row names have changed. Now, the hidden layer hidden layer you know bias and hidden layer bias value and nodes they have become the row names and the output layer nodes node has become the you know column name.

So, you can see bias that bias value for this particular output node and then we have this connection weight from node 3 to node 6 and node 4 to node 6 and node 5 to node 6. So, these are the connection weights and bias values for the model that we have just built.

Now, the final output values the values that we get from the output node, so this value. So, this value is also returned by the model by the function in our model object. So, this can be accessed using net dot not net dot result list somewhat dot what dollar net dot result will give us these core values. So, in this particular case we had just one node. So, we have just one value here. So, because this was for classification this was classification task.

So, these values can be compared with our cut off value. So, in this particular case we are taking 0.5 as the cut off value which is equivalent to the most proper class method where you know there are only two classes then 0.5 cut off value will actually implement that most proper class method. So, we will get these specific values using these scores. So, if you want to have a look at this scores also so we can look at this scores as well.

(Refer Slide Time: 16:28)

The screenshot shows the RStudio interface with the following details:

- Code Editor:** Displays R code for training a neural network. The code includes setting up input and hidden layers, defining connection weights, and training records. It uses the `modnet` function to create a neural network object and `modtrain` to train it.
- Console:** Shows the execution of the R code. The output includes the creation of a neural network object named `node5` with a value of  $-0.034636759934$ , and the resulting scores for each observation. The scores are listed as follows:

Observation	Score
1	0.5065925524
2	0.5065564247
3	0.5065737737
4	0.5065775296
5	0.5065896939
6	0.5065948789

So, if we run this you can see this is a list and we have just one node and 6 observations there. So, these are the scores. So, if we compare this with 0.5, we will get the values. Let us unname these column names, these names, dimension names and we will get this now we are creating a table with the scored you know this the predicted class and then

actual class and then the predicted value that is the you know we can say the probability scores and then predictors.

(Refer Slide Time: 17:09)

The screenshot shows the RStudio interface with the following details:

- Code Editor:** Displays R code for training a neural network. The code includes setting up hidden layers, defining connection weights, and creating a data frame to store predicted vs actual values along with predictor variables.
- Console:** Shows the execution of the R code, including the creation of a neural network object and the resulting data frame.
- Environment:** Shows the global environment with objects like `df` (a data frame with 6 observations and 3 variables), `mod` (a list of 13 elements), and `modtrainc` (a numeric vector).
- Data View:** Shows the contents of the `df` data frame.

```

65 # hidden layer to output layer connections
66 dimnames(mod$weights[[1]][[2]])= list(c("bias","node3","node4","node5"),
67                               c("node6:accept")); mod$weights[[1]][[2]]
68
69 # classify training records
70 modtrainc<-ifelse(mod$net.result[[1]][,1]>0.5, 1, 0); modtrainc
71 modtrainc<-unname(modtrainc)
72
73 data.frame("Predicted Class"=modtrainc, "Actual Class"=df$Acceptance,
74            "Predicted Value"=mod$net.result[[1]][,1], "fat"=df$FatScore,
75            "salt"=df$SaltScore)
76
77 # classification matrix
78 table("Actual Class"=df$Acceptance,
79        "Predicted Class"=factor(modtrainc, levels = c("0","1")))
80 # classification accuracy
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
626
627
627
628
629
629
630
631
632
633
634
635
635
636
637
637
638
639
639
640
641
642
643
643
644
645
645
646
647
647
648
649
649
650
651
652
653
653
654
655
655
656
657
657
658
659
659
660
661
662
662
663
663
664
665
665
666
667
667
668
669
669
670
671
671
672
673
673
674
675
675
676
677
677
678
679
679
680
681
681
682
683
683
684
685
685
686
687
687
688
689
689
690
691
691
692
693
693
694
695
695
696
697
697
698
699
699
700
701
701
702
703
703
704
705
705
706
707
707
708
709
709
710
711
711
712
713
713
714
715
715
716
717
717
718
719
719
720
721
721
722
723
723
724
725
725
726
727
727
728
729
729
730
731
731
732
733
733
734
735
735
736
737
737
738
739
739
740
741
741
742
743
743
744
745
745
746
747
747
748
749
749
750
751
751
752
753
753
754
755
755
756
757
757
758
759
759
760
761
761
762
763
763
764
765
765
766
767
767
768
769
769
770
771
771
772
773
773
774
775
775
776
777
777
778
779
779
780
781
781
782
783
783
784
785
785
786
787
787
788
789
789
790
791
791
792
793
793
794
795
795
796
797
797
798
799
799
800
801
801
802
803
803
804
805
805
806
807
807
808
809
809
810
811
811
812
813
813
814
815
815
816
817
817
818
819
819
820
821
821
822
823
823
824
825
825
826
827
827
828
829
829
830
831
831
832
833
833
834
835
835
836
837
837
838
839
839
840
841
841
842
843
843
844
845
845
846
847
847
848
849
849
850
851
851
852
853
853
854
855
855
856
857
857
858
859
859
860
861
861
862
863
863
864
865
865
866
867
867
868
869
869
870
871
871
872
873
873
874
875
875
876
877
877
878
879
879
880
881
881
882
883
883
884
885
885
886
887
887
888
889
889
890
891
891
892
893
893
894
895
895
896
897
897
898
899
899
900
901
901
902
903
903
904
905
905
906
907
907
908
909
909
910
911
911
912
913
913
914
915
915
916
917
917
918
919
919
920
921
921
922
923
923
924
925
925
926
927
927
928
929
929
930
931
931
932
933
933
934
935
935
936
937
937
938
939
939
940
941
941
942
943
943
944
945
945
946
947
947
948
949
949
950
951
951
952
953
953
954
955
955
956
957
957
958
959
959
960
961
961
962
963
963
964
965
965
966
967
967
968
969
969
970
971
971
972
973
973
974
975
975
976
977
977
978
979
979
980
981
981
982
983
983
984
985
985
986
987
987
988
989
989
990
991
991
992
993
993
994
995
995
996
997
997
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
162
```

(Refer Slide Time: 17:45)

The screenshot shows the RStudio interface with the following details:

- Code Editor:** Displays R code for training a neural network and calculating a classification matrix.
- Environment View:** Shows the global environment with variables like `df`, `modtrainc`, `modtraininc`, `Acceptance`, `FatScore`, and `mod`.
- Data View:** Shows the data frame `df` with 6 observations and 3 variables: Acceptance, FatScore, and modtraininc.
- Console:** Displays the R command history and the resulting classification matrix.
- Help:** A tooltip for the `neuralnet` function is visible.

```
70 modtrainc<-ifelse(modnet.result[[1]][,1]>0.5, 1, 0); modtrainc
71 modtrainc<-unname(modtrainc)
72
73 data.frame("Predicted Class":modtrainc, "Actual Class":df$Acceptance,
74             "Predicted Value":modnet.result[[1]][,1], "fat":df$FatScore,
75             "salt":df$SaltScore)
76
77 # Classification matrix
78 table("Actual Class":df$Acceptance,
79       "Predicted Class":factor(modtrainc, levels = c("0","1")))
80 # classification accuracy
81 mean(modtrainc==df$Acceptance)
82 # Misclassification error
83 mean(modtrainc!=df$Acceptance)
84
85 # neuralnet diagram
86
```

```
> modtrainc<-unname(modtrainc)
> data.frame("Predicted Class":modtrainc, "Actual Class":df$Acceptance,
+             "Predicted Value":modnet.result[[1]][,1], "fat":df$FatScore,
+             "salt":df$SaltScore)
> Predicted.Class Actual.Class Predicted.value fat salt
1          1           1   0.5065925524 0.2  0.9
2          1           0   0.5065564247 0.1  0.1
3          1           0   0.5065737737 0.2  0.4
4          1           0   0.5065775296 0.2  0.5
5          1           1   0.5065896939 0.4  0.5
6          1           1   0.5065948789 0.3  0.8
```

So, in this case as you can see the predicted classes all the observation have been classified as class 1. So, therefore, we need to make certain changes in this particular code that we have been using for the other techniques because there will not be any values for level 0.

So, therefore, we need to specify that explicitly, so that that actually comes in the classification matrix that we want to generate here. So, you can see that `modtrainc` I am converting it into a factor variable and then giving these two levels. So, that even if there are no observations being predicted as belonging to class 0 is still that particular you know column in the classification metric would be displayed; so let us run this.

(Refer Slide Time: 18:33)

The screenshot shows the RStudio interface with several windows open:

- Code Editor:** Displays R code for training a neural network and calculating classification metrics.
- Environment:** Shows the global environment with objects like `df`, `modtrainc`, and `modtrainnc`.
- Data View:** Shows the structure of the `df` dataset.
- Plots:** Displays a neural network diagram.
- Packages:** Lists available packages.
- Help:** Provides help for the `neuralnet` package.
- Viewer:** Shows the results of the neural network training.

**Code in the Code Editor:**

```
modtrainc<-felmnet(result[[1]][,1]>0.5, 1, 0); modtrainc  
modtrainc$unname(modtrainc)  
  
data.frame("Predicted Class"=modtrainc, "Actual Class"=df$Acceptance,  
          "Predicted Value"=modnet.result[[1]][,1], "fat"=df$Fatscore,  
          "salt"=df$SaltScore)  
  
# Classification matrix  
table("Actual class"=df$Acceptance,  
      "Predicted class"=factor(modtrainc, levels = c("0","1")))  
# classification accuracy  
mean(modtrainc==df$Acceptance)  
# Misclassification error  
mean(modtrainc!=df$Acceptance)  
  
# Network diagram  
modtrainc  
# (Top Level) #  
  
R Script
```

**Console:**

```
curvlinet(formula, data, hidden = 1, thresh=0.1,  
          stepmax = 1e+05, rep = 1, startwts =  
          learningwts, limit = NULL,  
          learningrate.factor = list(minus =  
          learningrate$minus, plus = learningrate$plus),  
          llikfun = "ll", lliksign = "+",  
          llikstep = 1000, algorithm = "logis-  
          reg", icl = "sic", act.icl = "logis-  
          linear", output = TRUE, exclude = NA,  
          constraint.weights = NULL, likelihood =  
          loglik, maxit = 1000, nstart = 1)
```

**Arguments:**

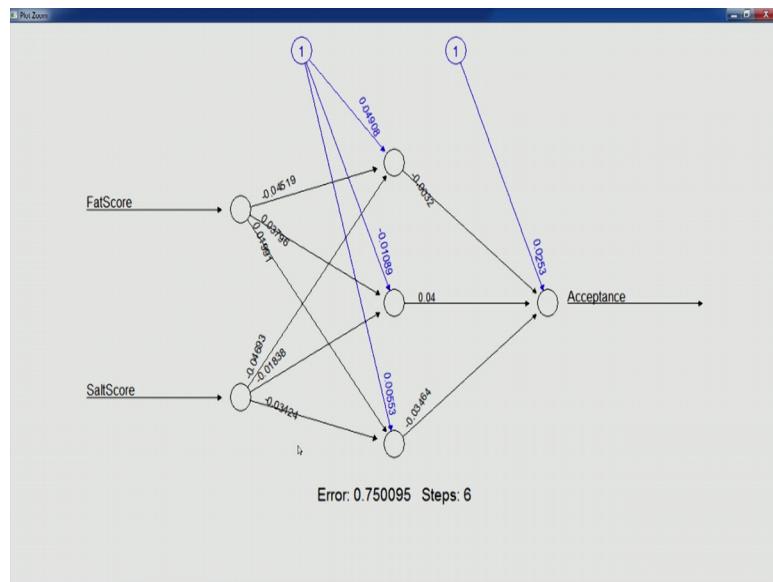
```
formula  
a symbolic description of
```

So, you can see this column I was talking about. So, no observation, but it is being displayed because of this change in the code that we have done.

We had used just the modtrainc directly this column would have gone. So, we can see that 3 values in the diagonal element 3 observation have been correctly classified and 3 observations are incorrectly classified. So, all the observation belong to you know you know in class 0 I have been incorrectly classified as class 1. So, the classification accuracy an error are also obvious 0.5 in this case.

We want to have a look at the network diagram now using R. So, this is the function plot and we have to pass the this neural network object mod here and we will get the diagram.

(Refer Slide Time: 19:27)

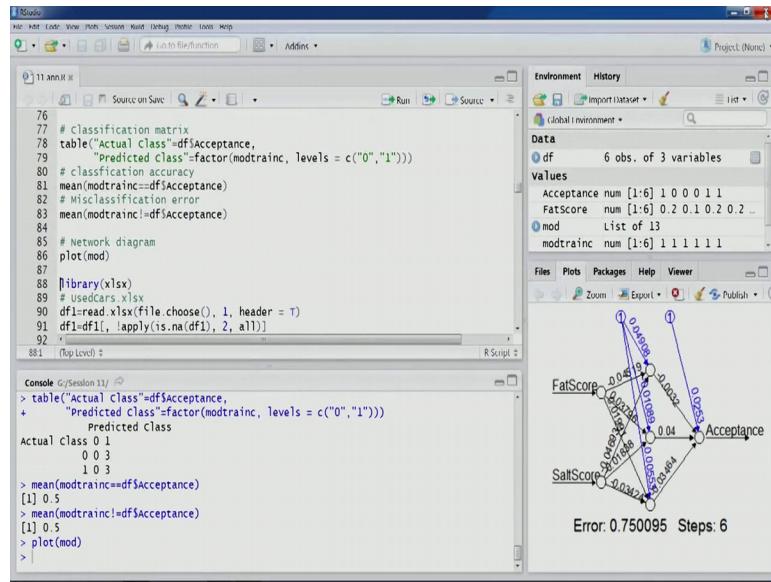


So, you can see here, so you can see here that this is the neural network diagram that has been prepared by this function plot.

So, you can see fatscore, saltscore these are the two predictors corresponding nodes so you can see here and from each node you can see the values the weights from both these nodes and you can see the bias values also here. So, 3 bias values corresponding to 3 hidden layer nodes here and then we have one bias node for the output node and 3 connection weights for the output node and then we have the, so this particular output node is corresponding to the acceptance that is our outcome variable.

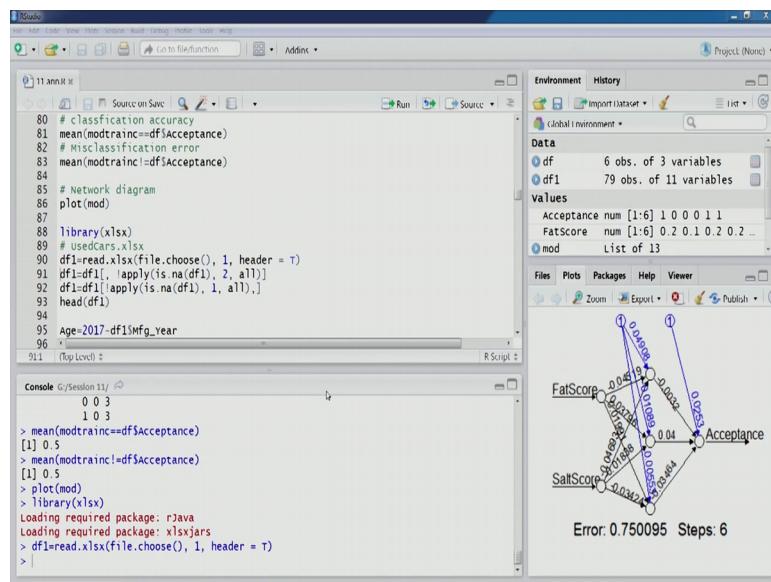
So, other details as you can see this was the error that is the sse value and the number of steps that were required to reach the conversion right to stop the network learning process. So, this is our model.

(Refer Slide Time: 20:31)



Now, so this example was you know we used small sample just 6 observations.

(Refer Slide Time: 20:42)

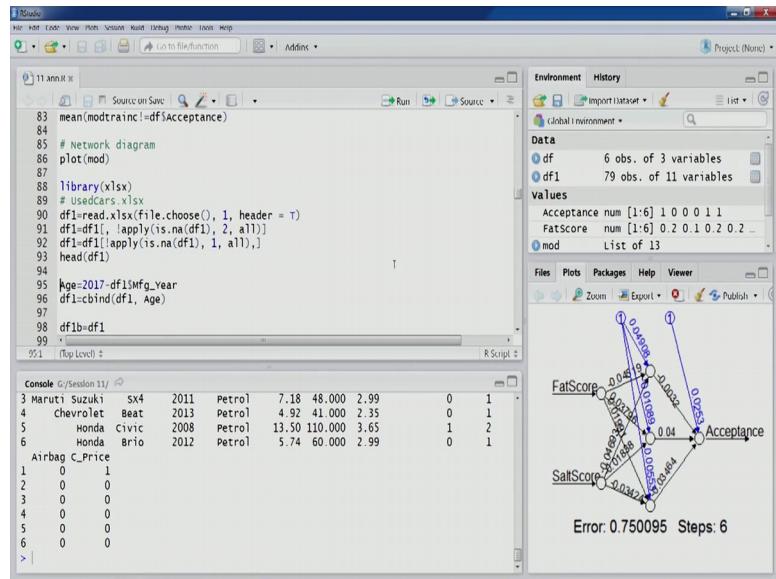


So, probably this was not an appropriate you know example an appropriate sample for us to understand you know build a good enough neural network model. So, what we will do we will use our used cars data set that we have been using in other techniques as well. So, this particular data set we are going to use to build a model with you know a slightly you know larger sample size.

So, because we would be importing data from excel file. So, let us load this package; there very now let us import this file with cars. So, you can see 79 observations of 11

variables; however, even for this dataset this data set is also small, but better than the previous example that we have used. So, let us remove na columns, na rows.

(Refer Slide Time: 21:38)



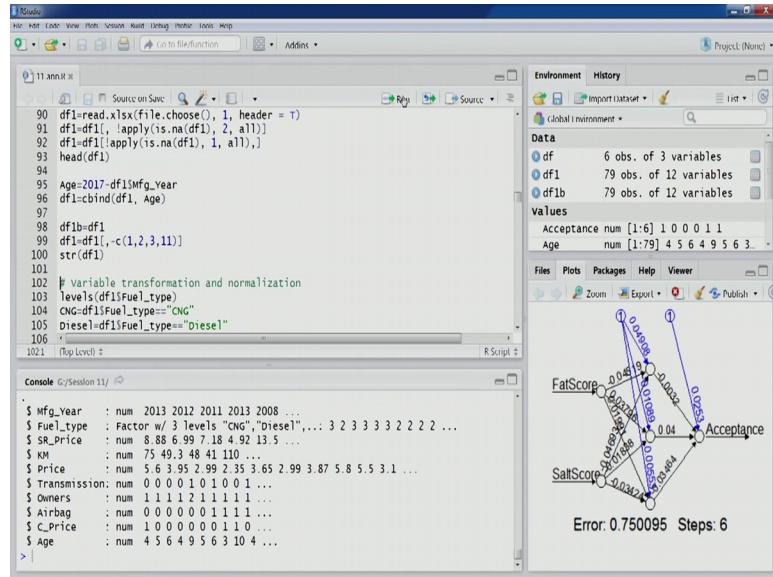
Let us look at the first 6 observations. So, these are the variables we are already familiar with this dataset brand so this is about the used car. So, we would like to build a prediction model to predict the value, offered price value of a used car. So, the variables that we have brand model manufacturing here Fuel type it is petrol, diesel or CNG then we have SR price that is show room price, that is the price when the that particular used the car was first purchased. Then we have kilometres the accumulated kilometres, then price the offer price of the used car in the current condition.

That is you know that we can see through the different variables. Transmission 0 or 1 that is manual or automatic, then the Owners previous number of 5 persons who have actually owned this car before this sale offer. So, that is there then the number of airbags and then we have another variable C underscore price; however, however we will not be using this particular variables C underscore price.

Now from the manufacturing year as we have been doing in other techniques also when we use the particular data set that we compute the age variables. So, that is more appropriate for our prediction model. So, let us compute this. Let us add it to data frame and let us also take a backup of this data frame.

Now, if we look at the variables, so now, we would like to you know get rid of certain columns certain variables.

(Refer Slide Time: 23:13)



For example, one is I think brand name yes; so brand name, model name and manufacturing you are no longer required and the last one that is as you can see this is this is actually C price we do not want. So, this is actually we have 12 variables this is the variable number 11.

So, we do not want this one as well, so because we are going to build a prediction model. So, let us cut it off these columns and now let us look at the variables that we have. So, we have now 79 observation 8 variables. So, these are the variables of interest to us. Now, what will do as we discussed in previous lectures that the neural network models perform much better when we have the scale of you know 0 to 1. So, if all the variables they are in this scale 0 to 1 then probably neural network model they perform they converge quickly and also the performance improves.

So, in this, particular data frame as we can see we have two categorical variable Fuel type and Transmission, so however, as we will see that neural network you know as we did in the previous exercise also the acceptance was categorical outcome variable, but we did not change it to change it into a factor variable; because the neural network function it does not allow us to do that conversion. So, all the computations are done internally in that particular function. So, we would be required to change some of these variables. So,

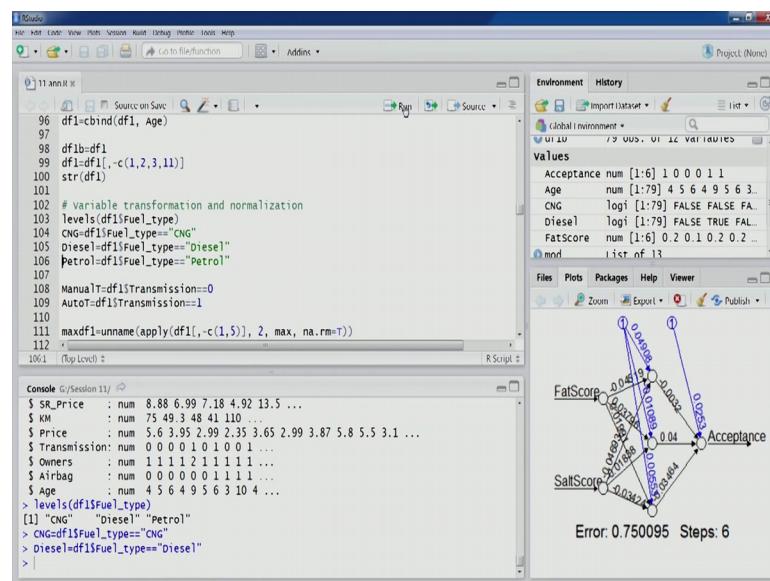
what we will do? We will see this is a one particular example in R environment where we have to explicitly create dummy variables.

So, dummy coding we have to do probably for the first time. So, we have been gone through so many techniques. So, typically we convert the variables into factor variable and the functions take care of the dummy conversion process and the model building later on however, in this particular case will have to explicitly do this. So, a fuel type we have 3 levels, let us check the level CNG, diesel, petrol. So, all these levels will look to convert them into dummy variables this is one way to achieve that you can see.

So, CNG if df1 dollar fuel type if it is CNG then because this logical operator we are using. So, therefore, if it is CNG then it will be true otherwise false. So, it could be a logical variable, you can see in the environment section CNG has been created logical you can see false false true. So, in the neural network function the values should be either the variables that use there should be the logical or numerical.

So, you are converting the factor variables the categorical variable into logical variables the logical dummy variables. So, now let us convert the diesel and then petrol.

(Refer Slide Time: 26:20)



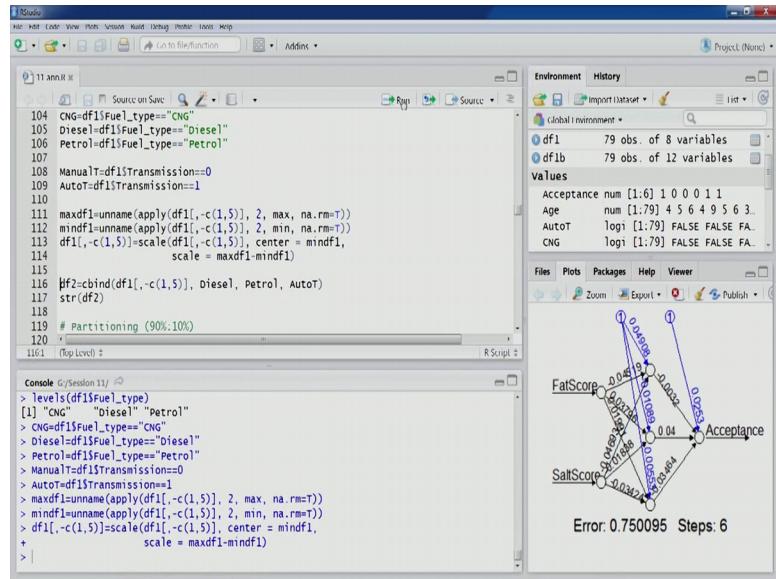
Then we have another variable transmission that is also factor variables; let us convert it into a logical or dummy variables. So, manual T and auto T however, as we understand that we will not be using all 3 all categories and one when we taken as the reference.

So, we would be no for fuel type we would be taking only two of the dummy variables in the modeling exercise and for transmission also just one of the dummy code in the modeling exercise. So, we will take diesel and petrol you know these two categories of Fuel type and the automatic transmission as the one category from transmission variable.

So, next process as we talked about as we discussed in previous lecture is we need to convert our numeric variable into you know bring them into a 0 1 scale. So, this is how we can do it. So, what first computation is we are trying to compute the max values for all the numeric variables; so you can see df1 data frame we have excluded a column number 1 and 5 which are the corresponding to fuel type and transmission. So, we left with only numeric variables here and then we are trying to compute max value for all these variables. And then in the next line we are trying to compute the min value for all these numeric variables. So, let us execute these two lines, and you would see that in max d of one here in the environment section we have 6 values because we have you know 6 numeric variable and again min df1, 6 values because we have 6 numeric variables.

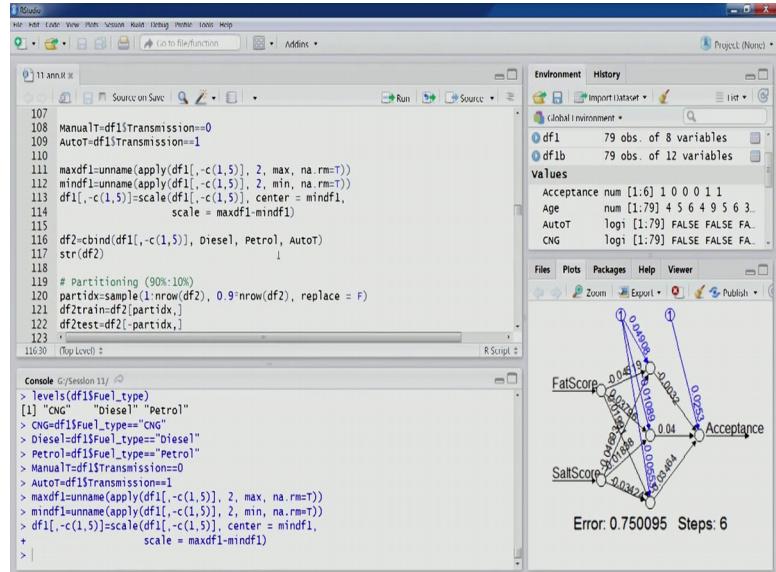
So, the max and min value have been computed. Now, we can use these values in the scale function. So, this scale function we are going to use to normalize to a 0 1 scale. So, you can see centre is now min df1 and the scale is max df1 minus min df1; so the particular formulation that we discussed in the slides so that is how it can be done using R. So, scale function can be used and then we will store these values in the same columns.

(Refer Slide Time: 28:45)



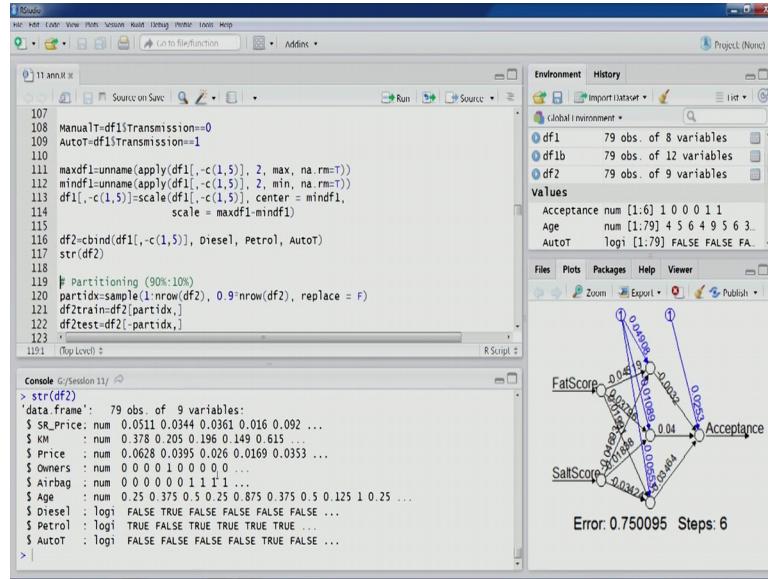
So, let us execute this code, now so we have scaled all these we have normalized all these numeric variables. So, let us add the dummy variables in and create a new data frame that we are going to use in our modeling exercise, so diesel, petrol and automatic transmission.

(Refer Slide Time: 29:02)



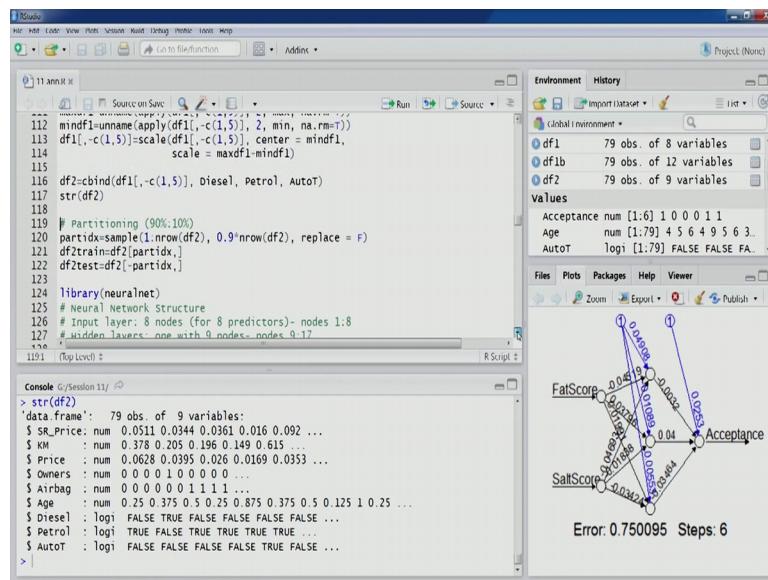
So, this was these are the variables that we are taking into our modeling exercise, let us create the data frame, let us look at the structure of this final frame.

(Refer Slide Time: 29:12)



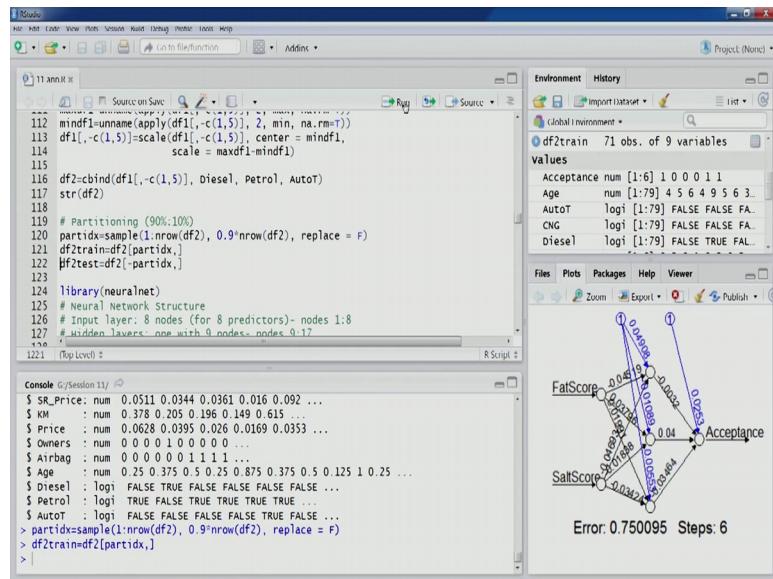
So, this is the frame df2 that data frame that we are going to use for our network model. So, you can see we have variable SR price, KM price, owners, airbag and age all have been scaled to 0 and 1 and you can see the values there in the structure output. And then we have 3 dummy variables which have been which have been taken as the logical variables here diesel, petrol and automatic transmission. So, once this is done we can go ahead and do our partitioning.

(Refer Slide Time: 29:47)



So, df2 is our data set now. So, we will take 90 percent of the values in the training partition and the remaining 10 percent of the values will be left for validation testing partition. So, let us create this. So, let us create training partition, then test partition.

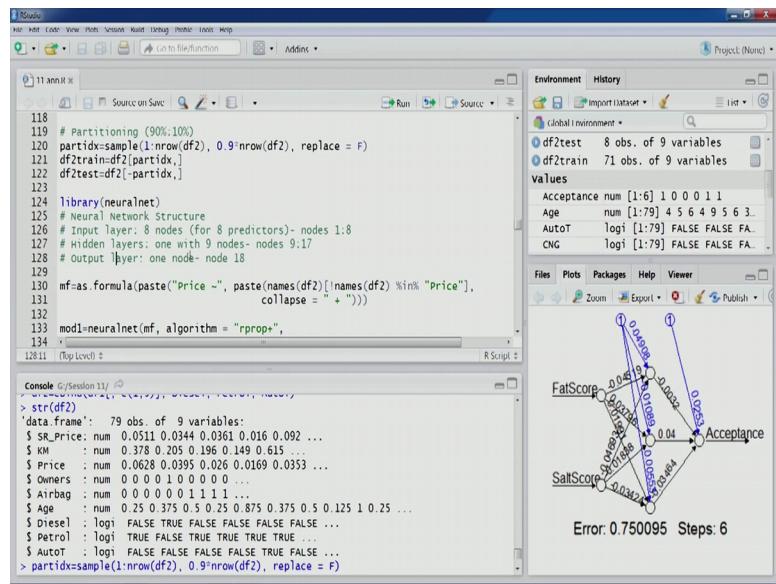
(Refer Slide Time: 30:08)



Now the same function the same packaged a neural net that we are going to use here; so neural network a structure that we have to decide right now if we look at the number of variables that we have. So, we have 9 variables, so one of them is the outcome variable that is price. So, that leaves us with 8 variables that is we have 8 predictors. So, therefore, in the input layer as we have been doing for previous examples. So, corresponding to 8 predictors we would like to have 8 nodes. So, that would cover nodes number 1 to 8; then hidden layer as we talked about that typically one hidden layer is sufficient to even model the complex relationships.

So, it will take just 1 hidden layer and you know we will take you know we will have 9 nodes, so 1 more than the number of predictors here. So, we had 8 predictors will let us take 9 nodes. So, of course, we can do experimentation with the number of nodes and even with the number of hidden layers. So, that will cover us the nodes number 9 to 17 then we will have a just 1 node in the output layer that is node number 18.

(Refer Slide Time: 31:24)



So, with this network structure we can go head and we can build our model and see the performance. Of course, after experimentation we can try out different candidate models as well and then finally, select one.

So, we will stop here and we will continue model access model building exercise for this particular dataset in the next lecture.

Thank you.