Time Series Modelling and Forecasting with Applications in R

Prof. Sudeep Bapat

Shailesh J. Mehta School of Management

Indian Institute of Technology Bombay

Department of Applied Statistics/Finance

Week 12

Lecture 60: Practical Session in R - 12

Hello all, welcome to this course on time series modeling and forecasting using R. So now we're sitting into the very last session of this entire course and then the course has culminated into its own journey starting with let's say discussing some time series applications and time series models, auto regression, moving average, ARMA, ARIMA, SARIMA. Then we talked about some other advanced models, then we talked about Fourier transformations. Then non-linear time series models, then machine learning integration is the topic here for this week where we are sitting on right now. And then today's session will be entirely devoted to practical session and hence we open the R session also.

And now the whole idea is to sort of implement some of the ML techniques we have seen over the course of this week, let us say over the last four sessions and try to integrate that using say simulated data from some ML or some time series models or bringing some practical data set, let us say air passengers data or maybe some other data and so on and so forth. And the whole idea is to implement some of the ML models on the time series applications and then see that how the forecasts are looking like or where exactly can you tweak the hyper parameters or change something in between and so on and so forth, okay. So before we begin, as always, you require a lot of packages for this particular code here. And then obviously I have installed each one of them.

But again, if you have not installed in the package, then again you have to go up here and click on tools and then install packages and then simply type the name here. For example, tidyverse, caret, metrics, etc. And once you type the name and again ensure that this is check here in the drop down menu the repository option and then again you may want to check this box here that install the dependencies and then click on install basically. And

obviously once you have installed you have to again make sure that the packages are brought in inside the R environment by using library command.

So initially for the very first exercise you require these three packages so tidyverse, caret and then matrix. So again I will ensure that we are bringing them inside the R environment. By running each and every row here. So, library tidy verse, then library caret, and then library matrix. So, again remember one thing that many of the times what happens is if you are using some inbuilt data set or the real data set, that real data set might not be available without a package.

Or the other way to put it is that you require some R package to load the real data set itself. For example, there could be some real data set inside the caret package or inside the matrix package etc. And for that reason you do first install the package. So, initially what we will do is we will try to run some of the ML modules on a simulated time series data set. So, again as always set dot seed some random number let us say 1, 2, 3 or could be any random number right and then sample size.

So, let us say you fix the sample size to be 300. So, n is 300 here. And this is exactly what the simulated time series looks like. So, let me run this line and then time series in our case is nothing but 20. So, 20 is a constant plus 5 into sine of a sequence from 0 to 10 into pi.

And how many observations do we want? We want n observations plus some random error. So, again, if you remember, how do we generate a random error? So, random error can be easily generated using R norm. So, if you assume that the distribution of the error is normal.

And let us say the mean is 0, the standard deviation is 1, and how many elements do we want in the random error vector? We want n, which is 300 here. So, essentially, what we are doing is we are generating a simulated time series by having a sinusoidal component in it using a sine command plus the random error component here. And, of course, at any point, if you want to find out what exactly the observations are, you can always typing the name of the dataset generated, so time underscore series in this case, and then it would sort of throw you the number of observations which are simulated from that time series here in the console. And since the sample size was 300, you can see that it is generating 300 values using this simulated time series.

But again, one should understand that what exactly are the or rather what exactly is the time series we are simulating from. So, again, just to repeat: 20 is the intercept, or rather the global mean, and 5 is the slope, or 5 is the constant multiplied by a sinusoidal component plus a random error. So, this is the very first simulated time series we are working with. Okay. Okay, so once you simulate the time series using the structure, then you have to create a data frame.

So data frame, why exactly? Because you want the values of the time series generated in one column and simply the index in another column. So time is nothing but 1 to n, so rather 1 to 300. And as a second column, we want the values generated which are here below, given below the console as the second column. So, we will create the simple data frame using data.frame command and then the first column is 1 to n, second value is nothing but the time series value that we have generated.

Now, we have to do some feature engineering. Again, if you remember for invoking or rather deploying any ML technique or ML model, we have to first control the number of features and sort of adjust the features. So, the entire exercise is called as feature engineering. So for that, what we'll do is we'll create some lag features. So this is a very standard approach as seen throughout this week that if you want to predict or forecast the current value, let's say yt, I can actually use its own lagged values as features.

So yt-1, yt-2, etc. So create some lagged features. Let's say you have function of... So I'll create a very simple R code. Now again, I'm not going to detail as to what this code means. But you are creating a simple function which takes the input as the data that we have generated and certain lags.

So, I will run this entire loop here, and this entire loop, once run, would give me some lag features of the current data points itself. Now, adding the three lag features. So, let us say you specify the number of lags to be 3. So, let us say the number of lags is 3, and then I will replace my data with the lag features which are generated using the above loop. So, now my feature set consists of the three lag features which are generated by the above loop.

So, hopefully, by this time, there should not be any problems, right? I mean, what do you mean by lags? How are you generating the lag features? Now again, how are you generating the lag features is not very easy. It is given by this entire loop here.

But then once you run the loop then essentially what we are having at the end of the day is we are having the lag features with us. And how many lags are we focusing on? We are only focusing on 3 lags here. So now, let me run this 24th line, which would give me the lag features. And now, the very important part: before you run any ML technique, you have to ensure that you split the data into training data and testing data.

So further, I have to actually tell R what the proportion of the training set and testing set should be. So here, can somebody answer what you think the proportion of the training set and testing set is? So here the proportion of the training set and testing set is 80% and 20% because what we are doing is we are randomly taking the n rows of our data and multiplying or rather taking 80% of those rows as the training data and the remaining 20% would be testing data. So, training size is nothing but 0.8 multiplied by the number of rows of the data we have. Let me run this, and then the train data is nothing but the data from 1 to train size, and we require all the columns, of course. So, only 80 percent of the rows, but all the columns, of course, and the remaining goes in the testing set.

Make sense? And why is it required? So, as discussed in the last session, you have to split the data into a training set and a testing set because you have to run the ML model on the training set, or rather, the correct way to put it is that you train the ML model using the training set, then you find the predictions, and then you compare the predictions with the testing set and find the accuracy. So, the accuracy of the test set is what is important to any ML researcher. So, if the accuracy on the test set is high, then the ML technique has successfully been capable of capturing the non-linearity in the data set.

And then again, down the line, we will see that we will do this entire similar exercise a number of times. A number of times means in each category we will look at today. So now, let me run the test data. So now, once you run these three lines, we have the training data and the testing data with us. Now, train a linear regression model.

So, again we will start with a very basic sort of a linear regression model on the training data. So, here again we will invoke the LM command. So, LM stands for linear model. We will give it a name, let us say model. And we are basically telling R that it should be a linear regression between.

So, the response variable is value. Response variable is value. And the set of features or rather the set of inputs are all the features we have. And data is which data is specified that the data to focus on here should be only the training data and not the entire data. So you run a or rather train a linear regression model on the training data.

Let me run this. And just to see that what R has given us, I can run something like a summary of the regression model. And then it will give you the summary of the regression model. And then here you can clearly see that since we have all these features, right? So the first lag, second lag, third lag and the time.

So the small table gives you the estimates of all the coefficients along with the standard errors, the t-value, and the corresponding p-values. And from here, you can see that all the layers are—or rather happen to be—significant because the p-values are less than 0.05. And down below, it sort of gives you, let us say, the R-squared value, adjusted R-squared value, and all those things. And then again, I will say that this is a very typical summary that R gives us if you run any linear regression model on a dataset. Again, just to repeat one last time: you should not forget that you should specify the data to be training data and not the entire data or testing data, something like that.

So, once you train a linear regression model on the training data, the next immediate thing is to predict on the test set. So, train the ML model on the training set and do the prediction on the test set. So, here we will run using the standard command called predict—predict using the regression model here—and then we specify that the new data has to be test data. So, you have to compare the predictions on the test data with the actual test data and then find the accuracy of that. So, let me run the predictions line.

Here, and then this would give me the predictions on the test data. Now, eventually, how do you evaluate the model performance? So, I can find out some of the standard metrics—let us say MSE, root MSE, etcetera. So, in this case, I will find out the MSE. So, the value of MSE happens to be 1.716, roughly. And now, just to visualize how the linear regression is performed, I can have a simple plot using the ggplot command, right?

And then, along with all the specifics. So here, again, I am not going to detail what each and everything means, but this is the syntax of ggplot. So, I can specify the colors, right? Or, let us say, line width or what sort of points you want in the plot. Or do you want any axis labels?

Do you want a title for the plot? So, all these things would be controlled using the ggplot syntax. So again, my strong suggestion would be that if you are more keen on exploring ggplot and want to change, let us say, some of the pointers or some of the colors there, then you should study this chunk here in detail. So, just to show you how the regression is performed, I will run the plot of the actual data along with the predicted data. Let me zoom in.

So, this is exactly what the time series, or rather the simulated time series, was looking like. So, the blue line is the actual data, which is the simulated data, and the red line is the predicted data. And here, you can see that since this is not highly non-linear in nature—I mean, you have some locally linear parts in there— I can sort of fit a linear regression line and transform that accordingly to get a superior fit here, okay. So, this is the actual data, and this is the predicted data given by the linear regression technique, okay.

Now, the second thing is, how about KNN? So, can we implement KNN? So, I mean if you remember a couple of sessions back, we talked about KNN or KN nearest neighbors technique in detail or it was pretty thorough, right. So, can we, so rather than implementing a linear regression which cannot control for a very highly non-linear sort of a setting, can we sort of move from a linear ML model to a sort of a non-linear ML model which is KNN. But again, remember one thing: for implementing KNN, you might require some extra packages.

So, I will first install them and then load them. So, let us say tidyverse and then the class package. Okay, so again, the next exercise is we will simulate a time series—a slightly different one—and then try to run KNN on that. So, rather than running KNN on the same simulated series, I will simulate a slightly different time series this time. So, here, what we will do is simulate a time series with a sinusoidal component and, of course, a random pattern.

So, again, let us say set.seed() with some random numbers: 1, 2, 3. The number of observations we want is 500. So, n is 500, and then, what exactly is the time series in our case? So, the time series in our case is the first chunk, which is the sinusoidal pattern. So, you have this first sinusoidal pattern, which is given by that—where 10 is a constant, 5 is the slope, and sine of 5 multiplied by pi, and the number of observations is 200. So, what we are doing here is, out of the 500 observations, we are splitting them into 200 observations containing the first sinusoidal pattern.

Then, the next 150 observations we are throwing in a random pattern in between. So, you can see the rnorm command. So, the rnorm command will give you a random pattern distributed normally, of course. And the last 150 observations will put forward another slightly different sinusoidal pattern. So, the entire architecture of the simulated data is that you have one sinusoidal pattern in the beginning, a random component in the middle, and towards the end, you have another sinusoidal pattern.

And then you will see that both the sine patterns are slightly different, right? Because the second pattern is a cosine pattern instead. So, in this simulated series, you have two sinusoidal patterns: the first one is given by sine, and the second one is given by cosine. So, let me run this generated time series—or rather, the simulated time series—and then visualize the series. So, I will show you a plot of the simulated series, and then it should make sense also. But then, for that, we have to convert the simulated series into a data frame.

So, the first column is nothing but time 1 to n, and then the second column I will give a name as 'value,' which is nothing but the series being generated here. And then the actual plot looks like this. Let me zoom in and then try to explain very quickly. So, this is the generated time series that we just did. So again, if you see here, the first pattern is a sinusoidal pattern containing a sine component.

Then in between where you see this lots of fluctuations is a random pattern. And then right towards the end is a cosine pattern which is slightly shifted. Now, the whole idea is that can we sort of invoke a k nearest neighbor ML algorithm to sort of find out the k nearest neighbors and then to predict whether you have a sinusoidal pattern or whether you do not have a sinusoidal pattern or maybe some other characteristics and so on and so forth. So, we sort of play around with the k nearest neighbors technique using this simulated data that we generated. Okay, so again if you remember the steps involved in KNN, the second step was segment the time series into overlapping windows.

So, the first would be to specify into how many windows do you want to segment the series into. So, let us say window size would be 20 and then segments. So, again this is more from a syntax point of view that how do you divide the series into different segments. So, the entire loop would just exactly that. and then label the data.

So, again if you remember or you want to convert the dataset into supervised sort of a format or label dataset. So, label data means what that assume we know the sinusoidal patterns occur at specific intervals. So, let us say if you again see this pattern here and assuming that you know exactly as to where the sinusoidal patterns occur. So, let us say in the beginning or at the end. and not in the middle.

So similarly, you can sort of label the data. So if you have any observation falling in between, you can say that it is not a sinusoidal pattern. If a particular observation is found in the beginning or towards the end, then you can label that as a sinusoidal pattern and so on. Hence, what we will do is, so this labeling exercise, what we are doing is if the start

time is less than equal to 200, or if the start time is more than 351, then we will label it as a sinusoidal pattern, otherwise a random pattern.

So, the whole idea here is a classification problem. So, the second chunk of code we are trying to focus on here is more from a classification problem. So, how do you run the KNN or KNRS neighbors for a classification sort of a problem? And how exactly? So, what we are doing is we are sort of labeling each and every observation.

So, before the 200th point, or after the 351st point, it is a sinusoidal pattern which you see in the figure also and anything in between would be a random pattern. So, this is exactly how we will label the time series. So, if the start time is up to 200 or less than equal to 200 or if the start time exceeds 351, you label the observation as a sinusoidal or otherwise in between you label the observation as random. So, hopefully this making clear to all of you that.

So, before you run the KNN, you need to actually tell R—or rather the ML model or the KNN model—what each and every data observation means by labeling them, okay? So, let me run this labeling code here, or the chunk in between, and now before you run the KNN, you would again do the same exercise of splitting the data into training data, testing data, and so on. So, I will not repeat this one more time in detail. Let me simply run the lines.

So, set.seed, then train indices, train data, test data, etc. And lastly, apply the KNN for classification. Okay. And here you can see that, from a syntax point of view, for applying the KNN, I am using all these yardsticks. So, mean, variance, slope, etc.—all the training data—then applying them on the testing data and getting the labels out of the testing data, okay?

So, train x, train y, test x, test y, And now, if you specify one of the hyperparameters—remember, how many hyperparameters are there in the KNN model? So, we have two, right? So, we have to specify the number of neighbors or the value of k. So, here, just some random number we have chosen—that will fix the number of neighbors to be three, let us say. So, we only want to focus on three neighbors of the target variable.

So, my K value is 3 and then we run the prediction. So, predictions and then you have this inbuilt command in R called as KNN. Of course, coming from some of the packages you know earlier. For example, so KNN is a part of these two packages here. So, tidy words and class.

Okay. So again, just to repeat one more time that for getting any real data into R or for running any inbuilt commands in R, let us say KNN, predict, all of that, you might require to install some packages every time. So again, going down, so let us say you implement the KNN algorithm on the training data. You specify what the testing data is. and you want what the class, you specify what the class is and also you specify how many neighbors you want.

So, k is 3, my test data is nothing but test underscore x, my train data is nothing but train underscore x and so on and so forth. So, if we run this prediction line and then finally, how do you check for model accuracy? So, one can evaluate some of the accuracies here. So, in this case, Since the classification problem, I can bring in the idea of accuracy.

So, accuracy is nothing but the mean of predictions that are exactly equal to the test value. And in this case, the accuracy is pretty high. So, 0.989. And now, lastly, we can visualize the patterns along with their predictions. So, let me run the ggplot command one more time.

And let me zoom in and try to explain briefly what is going on here. So, now again, if you clearly see, the blue dots represent the actual data given by the sinusoidal components, and the red dots are nothing but the random components, right? And, by the way, all these are coming from the test data. So, here you can see pattern recognition using KNN. Whenever the KNN model finds a blue dot, it labels that as sinusoidal correctly.

And if it finds a red dot, it labels it as a random component. And this is exactly as per what the simulated time series looked like, if you remember, right? Obviously, apart maybe from this point here. So, KNN is labeling this point as a random point, but it may be a sinusoidal component. You do not know because it is right here, right?

But apart from that, you can see that the QN has performed very well. In fact, in labeling each and every observation in the test set. By the way, again just to repeat all these observations are in the test set and the idea of KN is to label either whether the point is sinusoidal or random. So, all the red dots have been correctly labeled as random, all the blue dots have been correctly labeled as sinusoidal. And the overall construction looks exactly similar to the simulated data set that we had earlier.

Okay, so now we will play around with one real data set, right, and then for that we require all these packages one more time. So, tidy verse, random forest, metrics, etc. And

again, which data set we will use? So, we will again use the air passengers data set. So, let me load the air passengers data set.

Let me ensure that the data set is numeric by using as dot numeric and create a data frame one more time. So, my time argument is one all the way up to the length of the data and my value argument is nothing but the actual time series observations in the real data. Now, again we have to do some feature engineering by bringing some lag features of the data set and then how many lags we want. So, let us say we want 12 lags right.

So, we specify that, and then we again sort of replace the data value with the lag features. So, data is nothing but creating lag features. And then we will input data comma lands here. And again, the exact same exercise is splitting the data into training set and testing set in the proportion of 80% and 20%. Now, the model we are training the training set on is a random forest.

So, we will implement a random forest on the air passengers dataset. And then again, this is an inbuilt function in R, obviously, once you install the above packages, which is called random forest. And this is more like a syntax aspect of the random forest. So, we specify what the model is. On the training set, and then we apply the predictions on the test set by running the predict command, and then again, as usual, we will find out the MSE values or RMSE values, etc.

So, in this case, the MSE value happens to be 3778.922, and lastly, if you want to visualize the actual values with the predicted values, then this is exactly how the random forest has performed. And here, you can clearly see that the random forest has not performed quite well, right? Because the blue line is the actual value, the red line is the predicted value, and you actually have scope for improvement here, right? Because the random forest is not capable of capturing the peaks, right? So, one can actually go for some deep learning ideas or some other advanced techniques or something like that.

And now one last idea is implementing the NNAR on a real data set and for that we require these two packages forecast and ggplot and again the same air passengers data and then by the way this is exactly how the air passenger data looked like and here we will fit a seasonal NNAR model to the data and again why seasonal because air passengers data is seasonal. So, we have two different kinds of NNAR models. So, one is non-seasonal, the other one is seasonal. So, you can invoke or bring in seasonality in the NNAR model also. By the way, NNAR stands for Neural Network Autoregress, like we talked about just in the last session.

So, we will try to fit one particular NNAR model on the air passages data. This is the summary of the model and the forecasted values. And now, visually how it works. pans out. So, again it turns out that the NNAR model is doing pretty well here.

So, the blue line is the forecast and the blue and then obviously the black line is the actual data, but then you can see that the forecast is sort of preserving the seasonality, it is preserving the trend etc. And again as always you can find out the accuracies of all these measures you will get. And let us say I can do a step ahead and then try to split the data into training set, testing set. So, earlier we did not do that, right? So, we blindly forecasted using an NPR.

But the same thing can be done by splitting the data into training set and testing set, which would be rather a more formal sort of an exercise that people do, right? So, let us see what we have. So, we will split the data set into at this time point. So, December of 1959 we see. So, up to that point we will have the training data and all the observations beyond December 1959 would be testing data.

And now we will fit the NNIR model on the training data and compare the forecast with the testing data. And this is exactly how the plot looks like. So, here if you see let me zoom in. So the same exercise, the only difference is rather than simply forecasting down the line, I am splitting the data into training set and testing set. So now the red line is the actual observation path and the blue line is the forecasted plot.

And here you can see that the NNAR again is doing a very good job since the actual data and the predicted data from the NNAR model are sort of matching here. And lastly, I can as always find out the accuracy and so on and so forth. So hopefully using this small exercise, we've covered quite a lot in the practical session and then talked about random forest, migration, NNAR. We brought in some real data sets like air passengers, etc. So hopefully over the last 12 weeks or rather 30 weeks,

30 hours, you might have some flavor of, let's say, time series in general, its vast applications, and so on and so forth—nonlinear time series, then Fourier transformation, spectral analysis, all the elementary models, machine learning for time series, right? And by this, I am pretty sure that if somebody is at least interested in exploring this further, then I have done a decent job in explaining the ideas in this entire course thus far. Thank you.