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 58: Other Machine Learning Models for Time Series

Hello all, welcome to this course on time series modeling and forecasting using R. This is the last week we are entering, and as seen in the last couple of sessions, the broader topic for the entire week is machine learning aspects, or rather, machine learning integration into time series applications. And again the first two sessions were more from an introductory perspective where we sort of saw the different ideas about how and exactly what are the different ML techniques that one can adopt and how do you sort of integrate each and every one into some sort of an application pertaining to time series data. The very first session, we sort of discussed a lot of different ML techniques. And then again, if you are not very confident about any of them, then maybe one has to really pick some elementary book on ML and then try to read further. Because this course is not meant for sort of developing the details of each and every ML technique.

All right. So the last week I've kept specifically for just to make you all understand the integration of ML techniques and then time series applications per se. And again, if you remember, just to give you a very short recap, in the very last session, we talked about particularly three different kinds of ML techniques, where we talked about, let us say, starting with linear regression, then we talked about random forest, and then support vector machines, etc. Now, in this session, we will talk about a few other ML models and then we will see what to do next. And then, like I said, even before, the session afterwards would be entirely based on neural networks ideas and the integration of neural networks into time series.

But coming back to this session, we will focus entirely on some other advanced ML models. Let us say, if you are moving from a linear regression setup to maybe a classification sort of a setup or let us say random forest, decision trees, SVM, clustering

or let us say k nearest neighbors. So, we will talk briefly about all these different ideas along with their advantages, limitations and a few examples just to make sure that all of you understand the idea about how exactly can one sort of integrate the thought process behind let us say a particular ML model and then a time series application per se. So, as you see in front of you the heading of this slide is few other ML models. And again, we start with the very famous sort of an ML model called as K nearest neighbors or in short KNN.

Now, again, I am sure that many of you, if you have some knowledge about ML, must have come across this KNN technique. So, let me write it down. So, in short, it is called KNN or rather K-nearest neighbors. Now, again, here the idea is that, let us say, if you have a set of inputs, the idea is to sort of find And then the idea is to sort of find the nearest neighbors pertaining to the target variable, right?

And then sort of create a cluster around that, try to estimate the parameters, and eventually forecast, okay? So, again, this slide is just an introduction to KNN. So, the K-nearest neighbors, or in short KNN, is a versatile machine learning algorithm that can be adapted for time series forecasting or classification, okay? Now, again, if you remember, just to give you a short recap, let us say if you pick any ML technique that has its use cases pertaining to both regression and classification problems. So, regression means that the dependent variable Yt need not be only a categorical variable; it can be a numerical variable also.

But when it comes to classification, necessarily Yt has to be a categorical variable. And then eventually, the classification algorithm under this umbrella of ML sorts of classifies all the different input variables into different categories. So, based on a set of input variables, it categorizes each one of them into different categories and accordingly gives the output. So again, on one hand, we have a regression kind of problem. Again, this is just a small recap before we begin with anything new.

And on the other hand, let us say Yt is binary or, let us say, categorical, then the class of problems is called classification sort of problems. So in short, KNN is a versatile ML algorithm which can be adapted for time series forecasting or classification. In time series tasks, KNN is primarily used to identify the most similar historical patterns, which are also called neighbors, and use their outcomes to predict future values. So, as discussed a very short while back, that the idea, the primary idea behind KNN is to identify a set of most similar historical patterns, and these similar historical patterns are called as

neighbors, the other terminology is neighbors, and use their outcomes to predict any of the future values. So, essentially what it does is it finds the k-nearest neighbors.

So, when we say the k-nearest neighbors pertaining to any single point, we have to actually find out some similar historical patterns around that single point and use their outcomes to predict future values—hence the name k-nearest neighbors. So, hopefully, the idea behind KNN should be clear before we proceed with, let us say, some of the examples and applications and so on. Okay, so what exactly are the steps in action? So, let us say if one has to implement KNN, be it along with the time series application or without a time series application, but in general, let us say if you are sitting in a time series course and we are discussing the integration of, let us say, KNN with time series application, we will talk about steps in action accordingly.

So, the very first step is to convert any time series to a supervised problem. So, what do you mean by that? So, again, if you remember vaguely in the last session, we talked about an idea that any ML algorithm or the entire umbrella of ML can be categorized into two different sets of problems. The one is supervised. So, again, just a short recap, and on the other hand, you have unsupervised, obviously.

Okay. And let's say regression, classification, KNN, then let's say SVM, decision trees, random forest. So all these fall under supervised category. Now again, what do you mean by supervised? Supervised means that whatever inputs you have.

or whatever incoming data points you have are sort of labeled and you exactly know that which data point means what, okay. And rather on the other hand, what do you mean by unsupervised? So, unsupervised means that the input variables or the data points in the input set are not labeled and one very famous example under unsupervised is clustering. Let us say k means clustering and so on and so forth, okay. But here if you want to sort of implement the KNN or KNN as neighbors and again of course with an integration

With a time series application, the very first step is to convert any time series to a supervised problem, which means that you should exactly know that which observation sits in which time point and what exactly it means. So essentially, you want to label each and every data point. So again just to recap, so convert the time series to a supervised problem and inside this the first major point, we have two sub points. So the first sub point is a univariate time series is sort of converted into a supervised learning format by creating some lab features. Now, again, remember one thing that if you want to

implement any ML algorithm, there has to be a set of input variables, which are again called as features.

So, the technical term in ML terminology is called as features. And again, you should remember that in the first and second sessions this week, we talked about an idea called as feature engineering. So, the whole idea about feature engineering is to pick the correct features and then adjust the features and then sort of control each and every feature so as to sort of implement them into the ML model. So, again, just to recap, a univariate time series is converted into a supervised learning format. So, supervised is important here by creating some lag features.

For example, let us say the idea is to predict yt. So, yt is what is the response for us. So, essentially, the idea or the goal is to predict yt. You can use the features which are its own lag values. So, let us say yt minus 1, yt minus 2, up to yt minus p, where p stands for the number of lags.

And again, as seen even in the last session or the one before that, this is a very standard technique that one does, right? So, the integration of ML and time series, so since you already have this set with you, right? So, let us say if you want to predict the current state yt, you obviously have its own historical data points, right? So, yt minus 1, yt minus 2, yt minus 3, etc. So, why not make use of them in sort of

getting and trying to, trying to, sort of assuming that they are features, basically. Okay? So, this is the first step. Then, the second step is similarity measure. So, what do you mean by that?

So, similarity measure means that, again, remember the broad idea about KNN is. So, again, probably I will illustrate very briefly. Let us say you want to predict at this point. So, essentially, the goal is to find the K nearest neighbors. So, let us say you have all these observations.

Okay? And you want to find a small subgroup such that it is sort of neighboring this target variable. So, probably this, right? So, any data point which falls inside the circle could be considered as a K-nearest neighbor for that bigger red dot, okay? So, hence, how do you sort of formally measure the distance of all these observations from the target variable?

There has to be some yardstick, isn't it? And this yardstick is called as a similarity measure. So, the similarity between time series segments is calculated typically using

some distance measure. So, let us say Euclidean distance or something like that. So, again just to give you an example here.

So, what we will essentially do is to find out the Euclidean distance between each and every observation with the target variable. And essentially whichever Euclidean distance is minimum will consider those observations in the k nearest neighbor sector. So hopefully this idea is clear. So obviously these points here, since being far away from the target variable, won't make it into the K-NN set. And essentially this whole exercise is called a similarity measure.

So one requires some sort of a similarity measure to sort of capture the minimum distances between all the observations and the target variable. So again, just to recap, the similarity between time series segments is calculated, typically using some Euclidean distance. And second step inside the similarity measure heading is that each time step or rather row is treated as a vector in a p-dimensional space. So, again each time step or row is treated as a vector in a p-dimensional space. Now, again why p-dimensional?

Because again, if you go back a slide, then essentially what you are taking as features is nothing but the last p lag values. So, if you are taking the last p lambda values and then, let us say, if you have observations for each one of them, then the entire set would be in p dimensions essentially. So, hopefully, the first two steps are clear: one should always convert a time series to a supervised problem. And then, the second step is one should always invoke some sort of a similarity measure to measure the Euclidean distances between all the time series segments along with the target variable. And whichever distance measures are minimum, you keep them in the k-nearest neighbors set.

And then, the third step is prediction. So, once you are through with, let us say, estimation—finding the k-nearest neighbors, then estimation—then the last step is prediction. So, for any given test point—so a test point is any given point or rather a target variable for which you want to predict in the future. So, for any given test point, the algorithm finds the k-nearest neighbors from the training set. Now, let me pause here for a second. In the first two sessions, I did not discuss a very, very important idea: if you are implementing any ML technique,

There is always this idea about splitting the dataset into a training set and a testing set. Now again, I am very sure that most of you must already know this. If you have some awareness about, let us say, machine learning or you know all the ML techniques in

general. But the primary idea is that we should split the dataset into a training set and a testing set. And again, why should we do this?

So, we should fit the ML model, or rather, we should run the ML model on the training set. And here, we sort of assume that the training set is slightly larger. So, one can split the dataset into 80 percent, 20 percent, 70 percent, 30 percent in this ratio. So, the training set is slightly larger. You run the ML model on the training set and then try to predict on the testing set.

So, you can find out the accuracies. Okay. So, let us say you reserve some part of the dataset for measuring the accuracy of that particular ML model. So, let us say you build, for example, regression or classification SVM, any ML model, or let us say KNN. Then you run the model on the training set.

Right. And then predict that. And then you sort of compare those predictions with the actual dataset in the testing set. And from the testing set, you get the testing set accuracy. So, hopefully this small exercise is clear that once you split the data set into training part and testing part, you run the ML model on the training part, then you sort of predict in the future and then you sort of compare the predictions along with the testing set and then find the accuracies by let us say taking any standard measures, let us say MSE or you know root mean square error and so on.

So, again coming back to prediction for a given test point. So, test point means what? So, any point in the testing set. So, for that point we have not trained the ML model yet. So, we only train the ML model on the training set and then we sort of compare the predictions on the testing set.

So, for any given test point, the algorithm finds the k nearest neighbors from the training set. So, let us say from the training set, you run the KNN technique and then you sort of find broadly the k nearest neighbors and then you have a single point with the test set or the testing set and you use the ML algorithm which is being fitted on the training set to find out the k nearest neighbors of the test point. So, hopefully this is clear, all right. Then the second point is that the output for the test point is computed as let us say the average for regression problems or the majority class for classification problems. So essentially what would be the final output?

So the output for that particular test point in the testing set is computed as either the average if the problem is from a regression context or there is an idea called as majority

class. So majority class is whichever category is is weighed the most as per the ML output. So, if you have a regression problem, then one can take the average. If you have a classification problem, then of course, one cannot take the average because y t is categorical.

I will discuss this a short while ago. So, there we have a concept called the majority class. So, the prediction contains these two subpoints. So, you pinpoint a test point in the testing set, then you run the algorithm on the training set, find the k-nearest neighbors pertaining to the test point corresponding to the training set. And how do you get the output of the test point?

So, the output of the test point is computed by simplifying the average for regression problems or by figuring out the majority class for classification problems. Now, what exactly are some of the key components when it comes to KNN? So, again, many of the key components would be similar to, let us say, some of the ideas we discussed in the last session pertaining to regression, SVM, random forest, etc. For example, feature engineering. So, again, feature engineering is important here.

So, let us say lag features represent past values as predictors for future values. So, generally speaking, One actually takes some large values of yt itself, let us say yt minus 1, yt minus 2, yt minus 3, etc. as predictors for its own future values. The second point is to optionally include some rolling statistics.

So again, if you vaguely remember from the last session, we discussed the idea of rolling statistics. And then, one very famous rolling statistic is a moving average, right? So optionally, one can include some moving averages or rolling statistics to capture the trends, right? So, one can actually include both these ideas as features. And then, this entire exercise about identifying the features, controlling the features, and then running the ML model is for a feature engineer.

Then, the second key component when it comes to running a KNN is hyperparameters, right? So, what exactly are the hyperparameters under KNN? So, the very first hyperparameter is obviously K, right? Now, again, K is a number, right? So, K can be, let us say, 10, 15, or 20.

So, if K is 10, what do you mean by that? So, you are looking at the 10 nearest neighbors, right? Or whenever K is 15, you are looking at—you are talking about taking the 15 nearest neighbors. Or if K is 20, then you are talking about taking the 20 nearest

neighbors to the test point, and so on. So, identifying or rather controlling the value of K is rather important, right?

I mean, you should not go too overboard with the value of K and K should not be even too minimal. Now, remember one very important idea when it comes to ML technique is that you have an idea about underfitting and overfitting. So, again one should not go too overboard and one should not be underboard as well. So, one should try to balance all the essential aspects. So, for example, identifying K or finding out K, one has to strike the balance somewhere.

So, again what do you mean by K? So, K is nothing but the number of neighbors to consider. So, should you have 10 nearest neighbors or 12 nearest neighbors, 20 nearest neighbors, 30 nearest neighbors, etc., And the second important hyperparameter is a distance metric. So, typically Euclidean distance, but you do have some other alternative distance measures, let us say Manhattan distance and so on and so forth, ok.

So, one has to control for these two hyperparameters. So, the first one is K, which stands for the number of neighbors to consider. Second one is a distance metric, typically Euclidean distance or its own alternatives like Manhattan distance, etc., ok. The third key component is normalization. So, time series data often needs to be normalized.

So, what do you mean by normalized? Normalized means scaled down to ensure that all the features contribute equally to the distance calculation. So, one should actually go ahead and do the normalization by scaling them. So, each feature should be scaled down or normalized. And then fourth one is model evaluation.

So, again you have a standard technique of model evaluations. One can use common metrics for regression. Let us say MSE or MAE, etc. And if you have a classification problem, one can use let us say accuracy, precision, recall, etc. Now, what are the advantages of KNN?

So, simplicity. So, easy to implement and interpret. Non-parametric. So, non-parametric is an important advantage of KNN. So, non-parametric means that no assumptions about the underlying data distribution.

So, one need not assume that the distribution is normal or exponential, something like that. And it handles non-linearity pretty well. So, it can capture non-linear patterns in data. So, again, all these ideas now are one step ahead of linear regression. So, linear

regression cannot handle non-linearity, obviously, and all the other advanced ML techniques are slowly moving from linearity towards non-linearity.

So, simplicity, being a non-parametric approach, and handling non-linearity are some of the advantages of the K-nearest neighbors technique. On the other hand, what would be the limitations of KNN? So, obviously, it is costly. So, it is computationally costly. So, it is high for large datasets, as all the distances need to be computed for each prediction.

So, if you have a very, very huge dataset, then you have to compute each and every distance. Let us see: Euclidean distance, Manhattan distance, whatever. But you have to compute all the distances, which is quite a large task. And then, the second point is the curse of dimensionality. So, what do you mean by that?

So, high-dimensional data can dilute the effectiveness of distance metrics. Hence, one should carefully select the number of lag features and normalize the data. So, carefully choosing the number of lag features and normalizing the dataset are some solutions toward controlling the curse of dimensionality. Thirdly, no explicit model. So, KNN is memory-based and does not create a model, making it less efficient for frequent prediction.

So, in KNN, one cannot see the model, of course, as it is a black-box technique. So, one cannot really control the model per se. So, no explicit model exists within KNN. So, one simple example regarding KNN—and this is a very famous example, pretty much used effectively by many people—is called pattern recognition. So, what do you mean by pattern recognition, and what essentially is the integration of pattern recognition with time series?

So, particularly, pattern recognition in time series involves identifying segments of a time series that match a given pattern. So, let us say if you have a pattern in mind or in front of you, and you want to find some segments from the time series dataset that match the given pattern you have or belong to a specific predefined category. So, some examples could include identifying recurring events, anomaly detection, or grouping similar subsequences, etc. So, the whole idea about pattern recognition is, let us say, if you have some pattern, right? So, let us say—I will give you a hypothetical example—if you have this pattern in mind.

And let us say if you have some time series which is running like this and I will give you an example let us say something like that and probably somewhere in between you see

that you can sort of spot the same pattern probably here. right, the one which is given here. So, this exercise is called as pattern recognition. So, the idea is that you split the time series into multiple subgroups and then run the KNN, identify the KNN nearest neighbors in some way and then try to identify that which subgroup or which category of this entire time series is sort of similar to the pattern in mind, correct. And the examples here are let us say identifying recurring events, anomaly detection or grouping similar subsequences, etc.,

For example, you can use the KNN to recognize specific patterns in stock price movement. So, similarly exactly to what we talked about in this example, let us say if you want to identify some recurring patterns in the behavior of a stock, right, then one can use KNN or let us say machine vibration data or EEG signals to detect anomalies or classify patterns, etc., So how do you do this? So a set of brief steps. So the first step is define a reference pattern.

So there always has to be some reference pattern. And again, if you go back a slide, then the reference pattern in our case was something like that, which I do. So this would be the reference pattern. And then you try to identify or simulate a specific pattern you want to detect. Let's say, for example, a sinusoidal wave or a spike or a flat trend, etc.,

Okay. So, there has to be some reference pattern, and then you sort of browse the entire time series observations or the data points and try to find out whichever segment of the time series resembles the reference pattern. Okay. And then segment the time series. The second step is to segment the time series, of course.

So, divide the series into overlapping or non-overlapping windows, which are also called sub-sequences, to capture the local patterns. Okay. So, again, just to give you an example, let us say you have this overall time series, which runs like this. And let us say, for some reason, the time series behaves like that. Now, what we will do is divide this time series into overlapping or non-overlapping segments.

So, let us say you have non-overlapping segments, probably something like this. So, let us say you divide the time series into 1, 2, 3, 4. So, let us say 4 segments. And then you sort of run the reference pattern; you have the reference pattern in mind. And then you see which segment looks like or rather captures the reference pattern very well.

So, in this example, you can see that the third segment here sort of resembles the reference pattern that we have, ok. So, once you segment the time series, then you sort of

capture the reference pattern, you bring the reference pattern and then see that whichever segment sort of resembles the reference pattern, ok. Then extract the features. So, of course, it is a KNN, you require some features. So, again for each segment, you compute the features.

So, features could be what? Let us say mean, variance, slopes, Fourier coefficient. So, represent the subsequences. So, there has to be some model.

So, even if the model cannot be written down using pen and paper, but then you require some input variables which are called as features. Then the next step is train the KNN. So, KNN is an ML technique, you want to train the ML model on the training set. So, use the label data where segments are classified as pattern found or no pattern. So, you sort of run the entire KNN exercise or KNN model and then whenever it sort of resembles the reference pattern, then it sort of triggers this idea that pattern found and

Or if it is not able to find the reference pattern, then we trigger this idea, which is no pattern. And then, lastly, test on unseen data. So, again, as discussed a short while back, once you divide the data into a training set and a testing set, you want to implement it exactly on the test set. So, apply the KNN to classify new subsequences as matching the pattern or not. So, briefly, these are the steps when it comes to KNN for pattern recognition.

Now, what exactly are some of the other applications? Let us say, identify abnormal patterns in machine sensor data, such as vibration or temperature fluctuations. Detect head-and-shoulders or cup-and-handle patterns in stock prices. Recognize specific patterns in ECG signals, such as arrhythmias. Or detect rainfall patterns in weather time series.

So, all these are some other applications when it comes to pattern recognition. And in each one of these applications, one can again implement KNN very easily. Now, again, this is a slightly different idea; it is called naive Bayes. So, naive Bayes is a classification problem. So, naive Bayes for time series.

So, we will discuss very briefly what you mean by naive Bayes. So, naive Bayes is a very simple probabilistic classification algorithm. So, one should understand that it is not a regression problem; it is a classification problem based on the Bayes theorem. So, again, I am very sure that you must have studied the Bayes theorem sometime back, even in

school, let us say early college days and so on. So, this naive Bayes idea sort of revolves around the Bayes theorem.

It assumes that the features, let us say example lag values and time series, etc. are conditionally independent given the class label. This makes naive Bayes computationally efficient and easy to implement even for time series tasks. So, just to give an example, let us say in time series applications, naive Bayes can be used to classify sequences or sub-sequences of data into some predefined categories. For example, let us say identifying if a given time segment belongs to a normal or an abnormal class, etc.

Okay, so the whole idea about naive Bayes—I just thought of putting this slide there to sort of extend a step ahead from K and N. And naive Bayes is particularly for classification problems. So, one cannot bring in any regression problem inside naive Bayes. And the whole idea revolves around the Bayes theorem, as it sort of assumes that the features are conditionally independent given the class label. So, the whole idea of the classification problem is to classify each and every point in the output set into different categories, all right.

So, just to give you a time series application one more time is that let us say if you want to identify if a given time segment belongs to a normal category or anomalous class, etc. Then in all these applications one can implement the night base, okay. So, hopefully by this time you have some understanding about different ML techniques, the application in time series, how to implement each one of them in time series and so on and so forth. Now, when we enter the next session, so the next session would be entirely devoted to neural networks for time series. So, we will shift from let us say a standard ML approach to more from a deep learning approach, let us say neural networks.

And then we will discuss one very famous model which is called as NNER. So, we will try to integrate the neural networks and auto regression. So, all of that in the next session. Thank you.