Twitter sentiment analysis

People emotions to how customers felt about the product

Which could help companies understand their customers better. What competitors are doing. Which could help companies understand what customers are talking about it

Steps:

* Extracting tweets using Twitter application
* Cleaning the tweets for further analysis
* Getting sentiment score for each tweet
* Segregating positive and negative tweets

Install the relevant packages that we need. To extract tweets from Twitter, we will need package ‘twitteR’.

‘Syuzhet’ package will be used for sentiment analysis; while ‘tm’ and ‘SnowballC’ packages are used for text mining and analysis.

#  Install Requried Packages

installed.packages("SnowballC")

installed.packages("tm")

installed.packages("twitteR")

installed.packages("syuzhet")

# Load Requried Packages

library("SnowballC")

library("tm")

library("twitteR")

library("syuzhet")

Extracting tweets using Twitter application

consumer\_key <- 'dyqqcbRL1qLFCCJN3xsPqjxUp'

consumer\_secret <- 'gzG4bDMBr51AlWKlWU9MBEdbbMg8YH9TwFjkxJ9FiSt3chGeG4'

access\_token <- '265838776-KnjLd1uvwvJ7bKNUUaHOS7fKb6xZ0UP08b3ZcviD'

access\_secret <- '1870oxlWjteyHkX7eUCGvVpcRIjFv1xhIZS6SkCPPGHkw'

setup\_twitter\_oauth(consumer\_key, consumer\_secret, access\_token, access\_secret)

**Cleaning the tweets for further analysis**

tweets.df <- twListToDF(tweets)

head(tweets.df)

head(tweets.df$text)

tweets.df2 <- gsub("http.\*","",tweets.df$text)

tweets.df2 <- gsub("https.\*","",tweets.df2)

tweets.df2 <- gsub("#.\*","",tweets.df2)

tweets.df2 <- gsub("@.\*","",tweets.df2)

### Getting sentiment score for each tweet

word.df <- as.vector(tweets.df2)

emotion.df <- get\_nrc\_sentiment(word.df)

emotion.df2 <- cbind(tweets.df2, emotion.df)

head(emotion.df2)

sent.value <- get\_sentiment(word.df)

most.positive <- word.df[sent.value == max(sent.value)]

most.positive

most.negative <- word.df[sent.value <= min(sent.value)]

most.negative

### ##POSITIVE

### sent.value <- get\_sentiment(word.df)

### most.positive <- word.df[sent.value == max(sent.value)]

### most.positive

### #NEGATIVE

### most.negative <- word.df[sent.value <= min(sent.value)]

### most.negative

### #Segregating positive and negative tweets

### positive.tweets <- word.df[sent.value > 0]

### head(positive.tweets)

### negative.tweets <- word.df[sent.value < 0]

### head(negative.tweets)

### 

### neutral.tweets <- word.df[sent.value == 0]

### head(neutral.tweets)

### # Alternate way to classify as Positive, Negative or Neutral tweets

### category\_senti <- ifelse(sent.value < 0, "Negative", ifelse(sent.value > 0, "Positive", "Neutral"))

### table(category\_senti)

## Time Series is the measure, or it is a metric which is measured over the regular time is called as Time Series. Time Series Analysis example are Financial, Stock prices, Weather data, Utility Studies and many more.

## Introduction to ARMA Time Series Modeling

ARMA models are commonly used in time series modeling. In ARMA model, AR stands for auto-regression and MA stands for moving average. If these words sound intimidating to you, worry not – I’ll simplify these concepts in next few minutes for you!

We will now develop a knack for these terms and understand the characteristics associated with these models. **But before we start, you should remember, AR or MA are not applicable on non-stationary series**.

In case you get a non stationary series, you first need to stationarize the series (by taking difference / transformation) and then choose from the available time series models.

First, I’ll explain each of these two models (AR & MA) individually. Next, we will look at the characteristics of these models.

### Difference between AR and MA models

The primary difference between an AR and MA model is based on the correlation between time series objects at different time points. The correlation between x(t) and x(t-n) for n > order of MA is always zero. This directly flows from the fact that covariance between x(t) and x(t-n) is zero for MA models (something which we refer from the example taken in the previous section). However, the correlation of x(t) and x(t-n) gradually declines with n becoming larger in the AR model. This difference gets exploited irrespective of having the AR model or MA model. The correlation plot can give us the order of MA model.

Here are my observations :

1. There is a trend component which grows the passenger year by year.

2. There looks to be a seasonal component which has a cycle less than 12 months.

3. The variance in the data keeps on increasing with time.

We know that we need to address two issues before we test stationary series. One, we need to remove unequal variances. We do this using log of the series. Two, we need to address the trend component. We do this by taking difference of the series. Now, let’s test the resultant series.

adf.test(diff(log(AirPassengers)), alternative="stationary", k=0)

Augmented Dickey-Fuller Test

data: diff(log(AirPassengers))

Dickey-Fuller = -9.6003, Lag order = 0,

p-value = 0.01

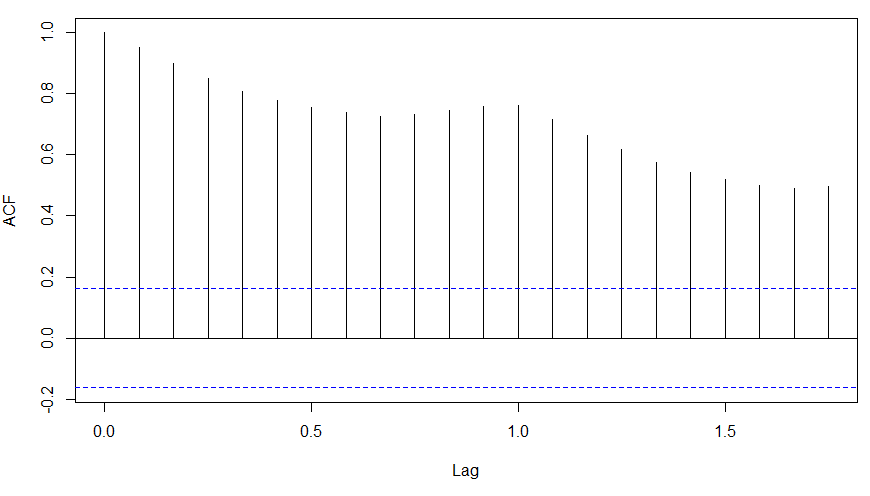
alternative hypothesis: stationary

We see that the series is stationary enough to do any kind of time series modelling.

Next step is to find the right parameters to be used in the ARIMA model. We already know that the ‘d’ component is 1 as we need 1 difference to make the series stationary. We do this using the Correlation plots. Following are the ACF plots for the series :

#ACF Plots

acf(log(AirPassengers))

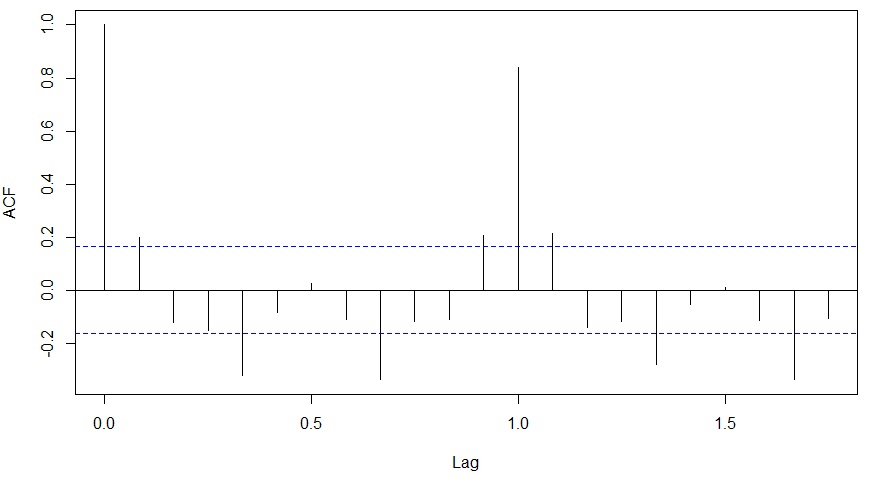
[](https://www.analyticsvidhya.com/wp-content/uploads/2015/02/ACF_original.png)

### What do you see in the chart shown above?

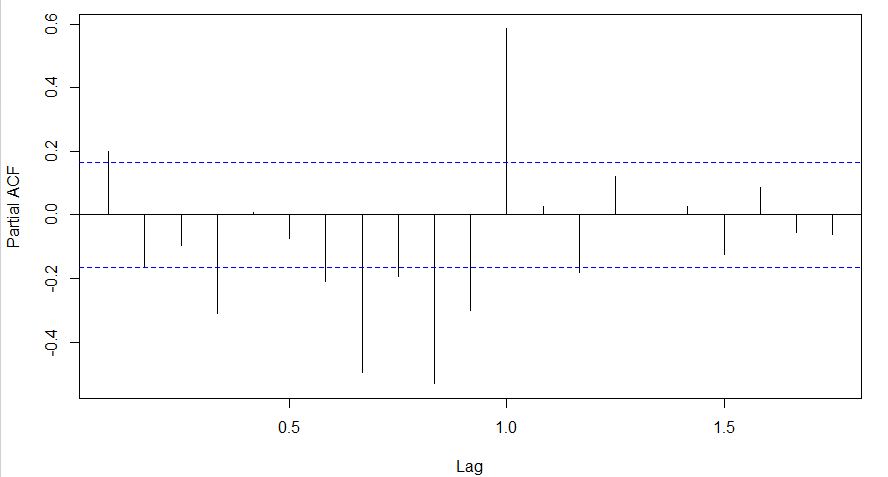
Clearly, the decay of ACF chart is very slow, which means that the population is not stationary. We have already discussed above that we now intend to regress on the difference of logs rather than log directly. Let’s see how ACF and PACF curve come out after regressing on the difference.

[stextbox id="grey"]

acf(diff(log(AirPassengers)))

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/02/ACF-diff.png)

pacf(diff(log(AirPassengers)))

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/02/PACF-diff.png)

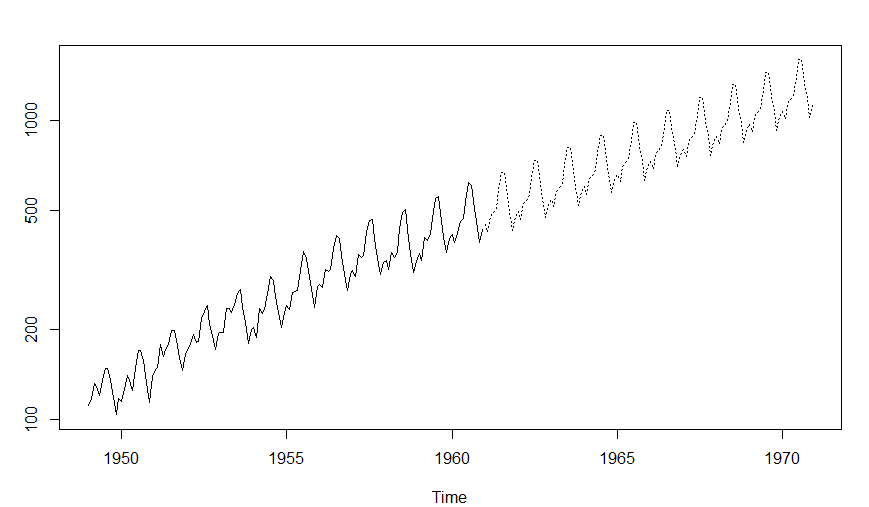
Clearly, ACF plot cuts off after the first lag. Hence, we understood that value of p should be 0 as the ACF is the curve getting a cut off. While value of q should be 1 or 2. After a few iterations, we found that (0,1,1) as (p,d,q) comes out to be the combination with least AIC and BIC.

Let’s fit an ARIMA model and predict the future 10 years. Also, we will try fitting in a seasonal component in the ARIMA formulation. Then, we will visualize the prediction along with the training data. You can use the following code to do the same :

(fit <- arima(log(AirPassengers), c(0, 1, 1),seasonal = list(order = c(0, 1, 1), period = 12)))

pred <- predict(fit, n.ahead = 10\*12)

ts.plot(AirPassengers,2.718^pred$pred, log = "y", lty = c(1,3))

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/02/predictions.png)

## Projects

Now, its time to take the plunge and actually play with some other real datasets. So are you ready to take on the challenge? Test the techniques discussed in this post and accelerate your learning in Time Series Analysis with the following Practice Problems:

**Survival analysis** corresponds to a set of statistical approaches used to investigate the time it takes for an event of interest to occur.

**Survival analysis** is used in a variety of field such as:

* *Cancer studies* for patients survival time analyses,
* *Sociology* for “event-history analysis”,
* and in *engineering* for “failure-time analysis”.

In cancer studies, typical research questions are like:

* What is the impact of certain clinical characteristics on patient’s survival
* What is the probability that an individual survives 3 years?
* Are there differences in survival between groups of patients?

### Survival time and type of events in cancer studies

There are different types of events, including:

* Relapse
* Progression
* Death

### Censoring

As mentioned above, survival analysis focuses on the expected duration of time until occurrence of an event of interest (relapse or death). However, the event may not be observed for some individuals within the study time period, producing the so-called censored observations.

Censoring may arise in the following ways:

1. a patient has not (yet) experienced the event of interest, such as relapse or death, within the study time period;
2. a patient is lost to follow-up during the study period;
3. a patient experiences a different event that makes further follow-up impossible.

This type of censoring, named right censoring, is handled in survival analysis.

### ?pbc

### #Fitting the survival model

### survival\_func=survfit(Surv(pbc$time,pbc$status == 2)~1)

### survival\_func

### #Plot the survival model

### plot(survival\_func)

### # Fit Cox Model

### Cox\_model = coxph(Surv(pbc$time,pbc$status==2) ~.,data=pbc)

### summary(Cox\_model)

### #Create a survival curve from the cox model

### Cox\_curve <- survfit(Cox\_model)

### plot(Cox\_curve)

### # install.packages("survival")

### # Loading the package

### library("survival")

### #Dataset description

### ?pbc

### #Fitting the survival model

### survival\_func=survfit(Surv(pbc$time,pbc$status == 2)~1)

### survival\_func

### #Plot the survival model

### plot(survival\_func)

### # Fit Cox Model

### Cox\_model = coxph(Surv(pbc$time,pbc$status==2) ~.,data=pbc)

### summary(Cox\_model)

### #Create a survival curve from the cox model

### Cox\_curve <- survfit(Cox\_model)

### plot(Cox\_curve)

### #Using the Ranger package for survival analysis

### #install.packages("ranger")

### library(ranger)

### #Drop rows with NA values

### pbc\_nadrop=pbc[complete.cases(pbc), ]

### #Fitting the random forest

### ranger\_model <- ranger(Surv(pbc\_nadrop$time,pbc\_nadrop$status==2) ~.,data=pbc\_nadrop,num.trees = 500, importance = "permutation",seed = 1)

### #Plot the death times

### plot(ranger\_model$unique.death.times,ranger\_model$survival[1,], type = "l", ylim = c(0,1),)

### #Get the variable importance

### data.frame(sort(ranger\_model$variable.importance,decreasing = TRUE))

### #Comparing models

### library(ggplot2)

### #Kaplan-Meier curve dataframe

### #Add a row of model name

### km <- rep("Kaplan Meier", length(survival\_func$time))

### #Create a dataframe

### km\_df <- data.frame(survival\_func$time,survival\_func$surv,km)

### #Rename the columns so they are same for all dataframes

### names(km\_df) <- c("Time","Surv","Model")

### #Cox model curve dataframe

### #Add a row of model name

### cox <- rep("Cox",length(Cox\_curve$time))

### #Create a dataframe

### cox\_df <- data.frame(Cox\_curve$time,Cox\_curve$surv,cox)

### #Rename the columns so they are same for all dataframes

### names(cox\_df) <- c("Time","Surv","Model")

### #Dataframe for ranger

### #Add a row of model name

### rf <- rep("Survival Forest",length(ranger\_model$unique.death.times))

### #Create a dataframe

### rf\_df <- data.frame(ranger\_model$unique.death.times,sapply(data.frame(ranger\_model$survival),mean),rf)

### #Rename the columns so they are same for all dataframes

### names(rf\_df) <- c("Time","Surv","Model")

### #Combine the results

### plot\_combo <- rbind(km\_df,cox\_df,rf\_df)

### #Make a ggplot

### plot\_gg <- ggplot(plot\_combo, aes(x = Time, y = Surv, color = Model))

### plot\_gg + geom\_line() + ggtitle("Comparison of Survival Curves")

### write.csv(mtcars,"C:\\Users\\vkumar15\\Desktop\\Desktop - Training\\Weekend\\Sunday\\mtcars.csv")

### install.packages("RMySQL")