Prophet_Exploration

paul

March 6, 2017

Contents

References:

- github Source Code
- Quick Start In R
- Wikipediatrend Vignette
  - used to scrape data from wikipedia pages.
- Prophet Function Documentation

Notes about this Package:

- This package is used to forcast data and is used by facebook to account for seasonal and holiday affects on the data.
- Backend is implemented in Stan
- Always take a data frame with 2 columns:
  - *ds:* a date/datetime column indicating time.
  - *y:* a column for the data to be forcasted. **Must be Numeric Values**
    - At least a year worth of data.
  - Other data like holidays, carry capacity.
- Returns a model object that can be processed using *predict* and *plot*.
- The *predict* function defaults to predicting a linear trend. When trying to log transform the *y* column, special attention needs to be taken to modify values of *0*.

General Steps:

1: Use *prophet* function to fit the historic data. 2: Use *make_future_dataframe* function to make dataframe with future dates for forecasting. 3: Forcast using *predict* function with both the historic and future data as parameters. 4: Call *plot* to plot the forcast (the historic data and the forecast)

Working Through Tutorial Online:

- ***Goal: Forecasting the time series of daily page views for the Wikipedia page for Peyton Manning._***
- Reading Data:

```
## Due to limited understanding of statics, all zeros are removed from the data set.

df = read.csv("./PeytonManningData.csv")%>% select(date:count)%>%
filter(count>0)%>%mutate(count=log(count))

##Beware, The names of the data frame column needs to be ds and y.
colnames(df)= c("ds","y")
```

- **Fitting Prediction:**

```
m=prophet(df)
```

```
## STAN OPTIMIZATION COMMAND (LBFGS)
## init = user
## save_iterations = 1
## init_alpha = 0.001
## tol_obj = 1e-012
## tol_grad = 1e-008
## tol_param = 1e-008
## tol_rel_obj = 10000
## tol_rel_grad = 1e+007
## history_size = 5
## seed = 590591393
## initial log joint probability = -8.52316
## Optimization terminated normally:
##   Convergence detected: relative gradient magnitude is below tolerance
```

```
names(m)
```

```
##  [1] "growth"                  "changepoints"
##  [3] "n.changepoints"          "yearly.seasonality"
##  [5] "weekly.seasonality"      "holidays"
##  [7] "seasonality.prior.scale" "changepoint.prior.scale"
##  [9] "holidays.prior.scale"    "mcmc.samples"
## [11] "interval.width"          "uncertainty.samples"
## [13] "start"                   "end"
## [15] "y.scale"                 "stan.fit"
## [17] "params"                  "history"
```

- *growth:* the growth model that is used to make the forecast. linear or logistic options are available.
- *changepoints:* which time values are changepoints.

- *n.changepoints:* total number of changepoints.
- *yearly.seasonality:* Boolean indicating if yearly seasonality is included.
- *weekly.seasonality:* Boolean indicating if weekly seasonality is included.
- *holidays:* a dataframe indicating holidays.
- *seasonality.prior.scale:* adjusting effect of seasonality prior. Higher scale setting, tend to increase the effect.
- *changepoint.prior.scale:* adjusting effect of changepoint prior. Higher scale setting, tend to increase the effect (increase is to increase flexibility of rate changes and allow more changepoints to be used.)
- *holidays.prior.scale:* adjusting effect of holidays prior. Higher scale setting, tend to increase the effect.
- *mcmc.samples:* Whether or not to perform the full bayesian inference with specified sample amount.
- *interval.width:* adjusting uncertainty intervals. refer to help.
- *uncertainty.samples:* Number of simulated draws used to estimate uncertainty intervals
- *start:* start date of historical data.
- *end:* end date of historical data.
- *y.scale:* maximum scaled y value.
- *stan.fit:* not too sure what this is.
- *params:* seems to be prediction parameters.
- *history:* historical/training data.
  - contains input data (ds,y,cap).
  - also contains scaled time, scaled y and scaled caps.

```
## produce a data frame specifying future dates to predict
future <- make_future_dataframe(m, periods = 365)
tail(future)
```

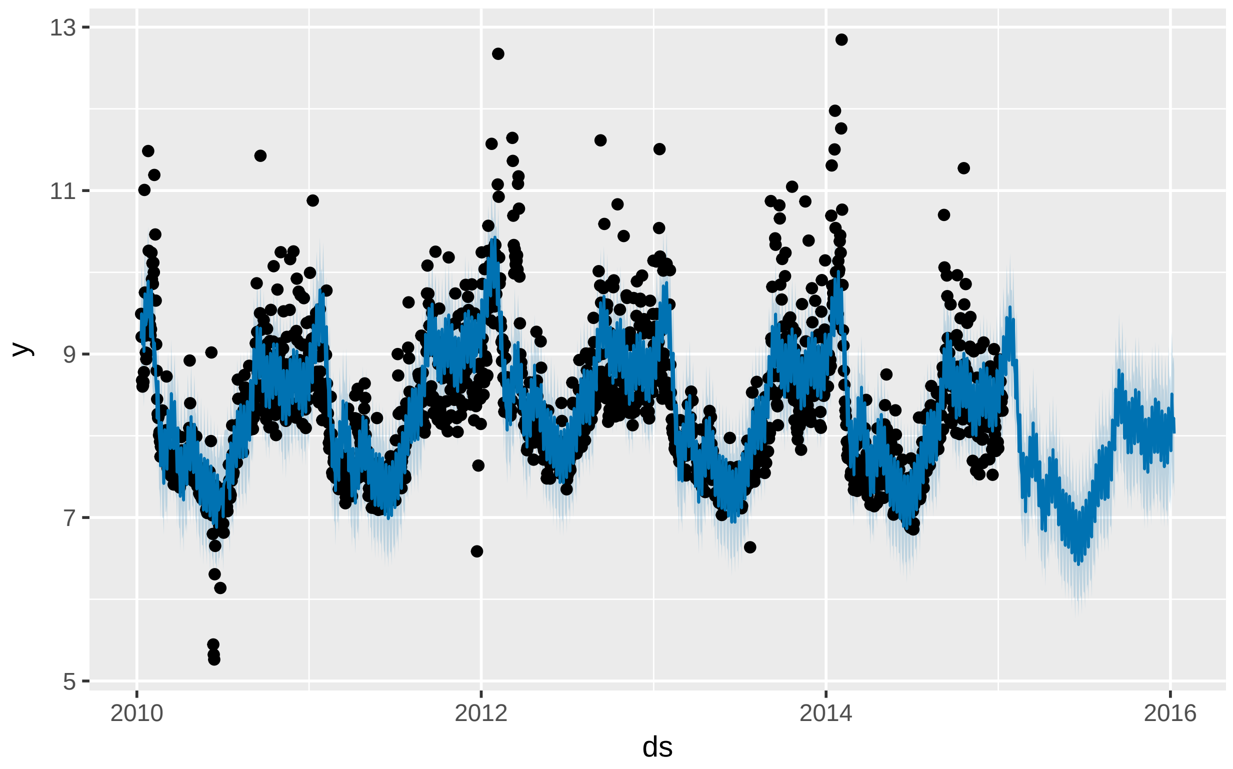
```
##              ds
## 2167 2016-01-04
## 2168 2016-01-05
## 2169 2016-01-06
## 2170 2016-01-07
## 2171 2016-01-08
## 2172 2016-01-09
```

- **Constructing future plot:**

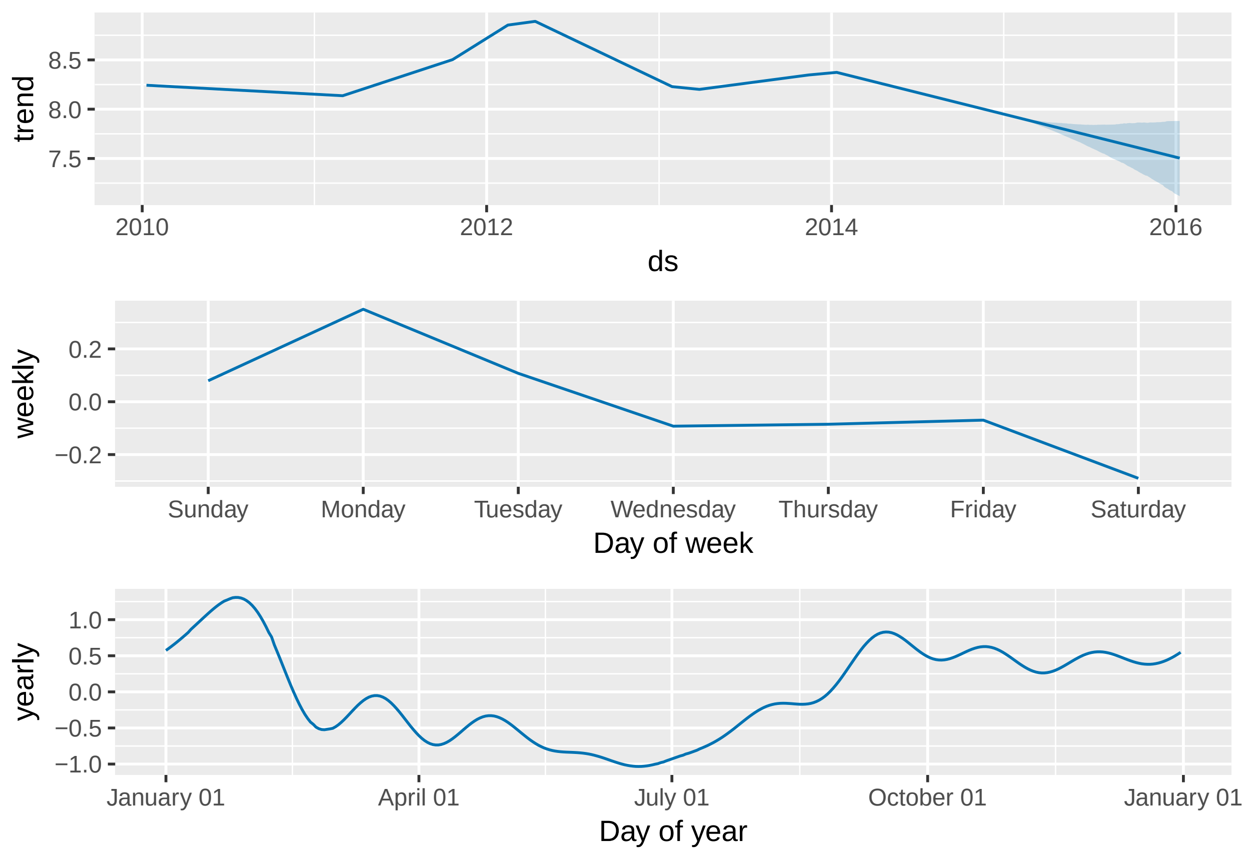
```
forecast <- predict(m, future)

#yhat contains the predicated information.
tail(forecast[c('ds', 'yhat', 'yhat_lower', 'yhat_upper')])
```

```
##              ds     yhat yhat_lower yhat_upper
## 2167 2016-01-04 8.512442   7.703854   9.273884
## 2168 2016-01-05 8.299222   7.522425   9.073750
## 2169 2016-01-06 8.129643   7.328667   8.872198
## 2170 2016-01-07 8.168003   7.417552   8.958897
## 2171 2016-01-08 8.215535   7.426623   8.955069
## 2172 2016-01-09 8.027930   7.247944   8.804147
```

```r
plot(m,forecast)
```



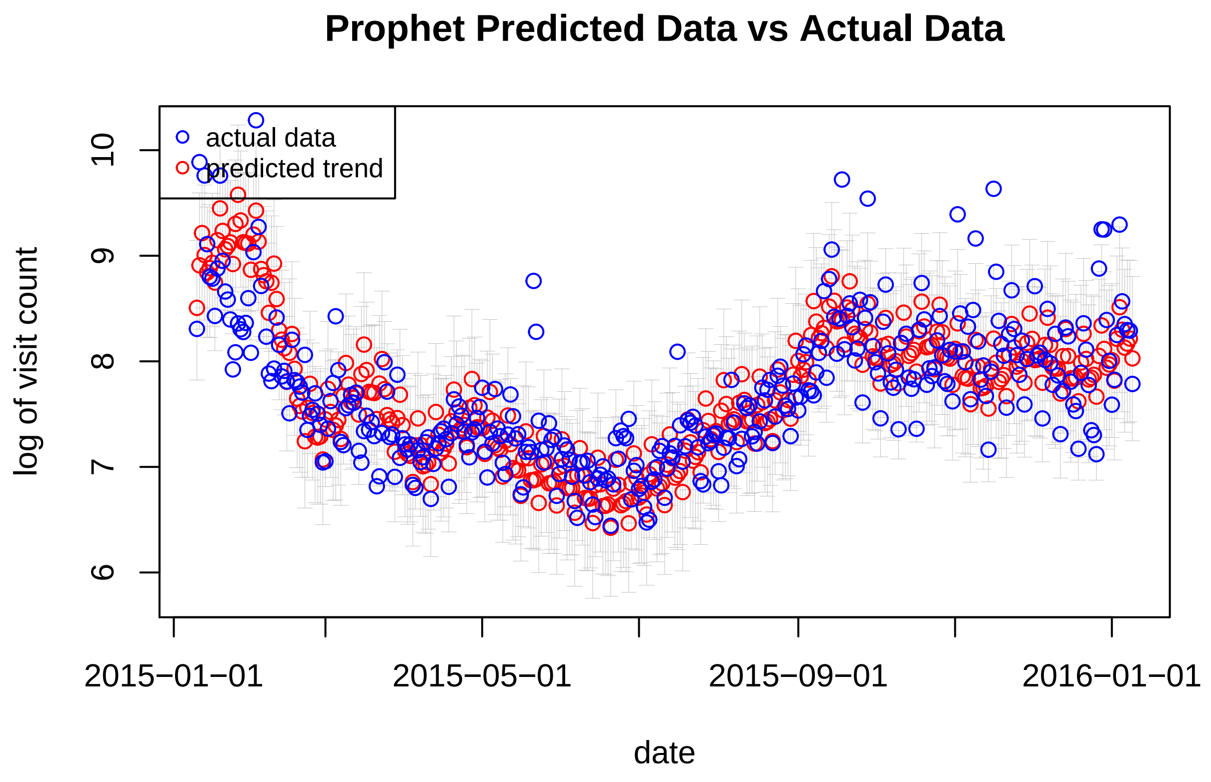
```r
## more detailed plot outlining where the data is broken down
## into: trend, weekly seasonality, and yearly seasonality

prophet_plot_components(m, forecast)
```



Comparing To Actual Data:

```
## Last 365 data point of forecast. The date for these data is needed
## to request the corresponding information from wikipedia
predicted_Data = tail(forecast[c("ds","yhat","yhat_lower","yhat_upper")],n=365)
start_Date = tail(forecast[["ds"]],n=365)[[1]]
end_Date = tail(forecast[["ds"]],n=365)[[365]]

# data = wikipediatrend:: wp_trend(page="Peyton_Manning",
# from = start_Date, to=end_Date)
# ##writing data:
#  if(!file.exists("actualData.csv"))
#  {
#      file.create("actualData.csv")
#  }
# write.table(data, "./actualData.csv",sep=",")

actual_Data = read.csv("./actualData.csv") %>% select(date:count)%>%
    filter(count>0)%>%mutate(count=log(count))
```

```
# plot(x=actual_Data[["date"]], y=actual_Data[["count"]],xlab="date",
# ylab="log of visit count",col="blue",main = "Prophet Trained Forcast
# vs Actual Data",type="l")
Hmisc::errbar(x=predicted_Data[["ds"]],y=predicted_Data[["yhat"]],
                   yplus=predicted_Data[["yhat_upper"]],yminus = predicted_Data[["yhat_lower"]],
                   lwd=0.1,pch=NA,errbar.col="gray",xlab="date",ylab="log of visit count",xaxt="n")

axis.Date(1, x=predicted_Data[["ds"]],format="%Y-%m-%d")
    points(x=predicted_Data[["ds"]],y=predicted_Data[["yhat"]], col="red")
    points(x=as.Date(actual_Data[["date"]],format="%Y-%m-%d"),
           y=actual_Data[["count"]],col="blue")

    title(main="Prophet Predicted Data vs Actual Data")

    legend("topleft",legend=c("actual data","predicted trend"),
           col=c("blue","red"),pch=1,cex=0.8)
```



Forecast Growth Models:

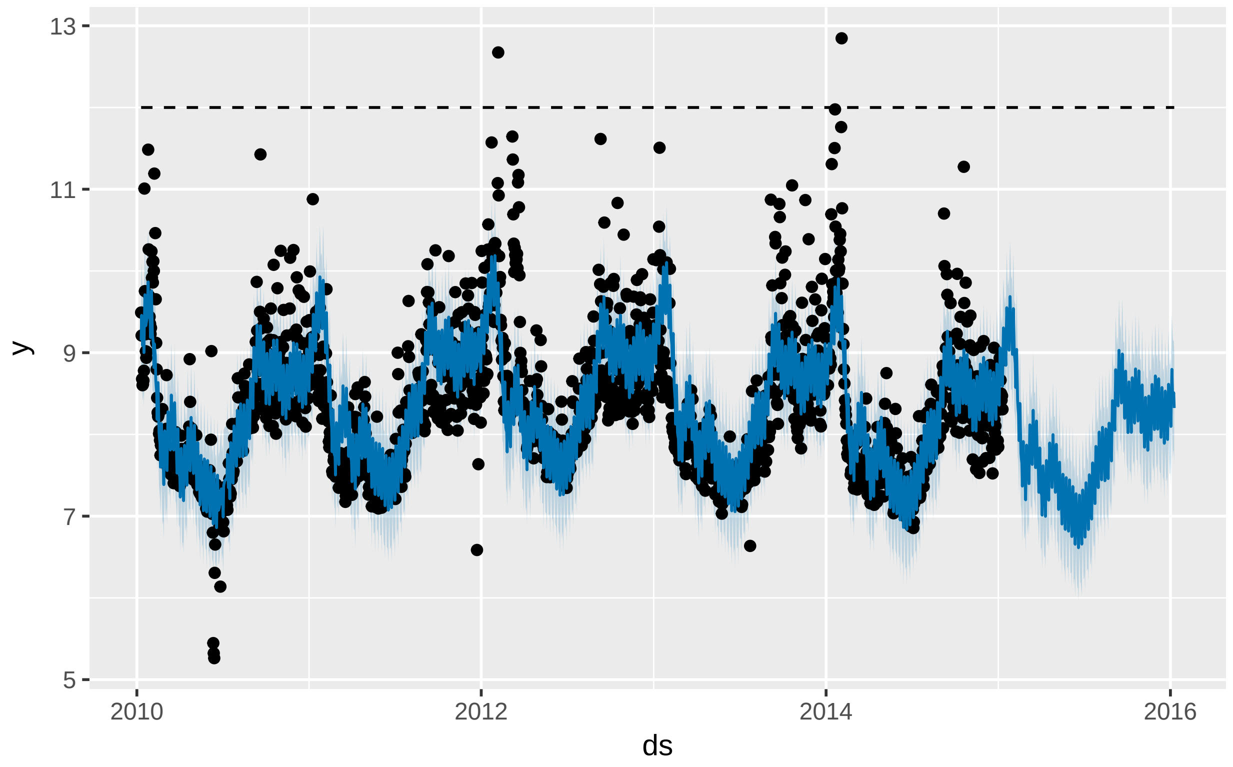
Forecasting

- By default, the predications are made using a linear model or a log function.
- The user can specify carry capacity in a column under the data, the maximum achievable amount. This must be done for logistic growth.
  - the carry capacity for each row can be defined.
- Changing the predications and setting carry capacity:

```r
    library(prophet);
    library(dplyr)

    df$cap <- 12##would be calculated using data normally
    m1 <- prophet(df,growth = 'logistic')
```

```
## STAN OPTIMIZATION COMMAND (LBFGS)
## init = user
## save_iterations = 1
## init_alpha = 0.001
## tol_obj = 1e-012
## tol_grad = 1e-008
## tol_param = 1e-008
## tol_rel_obj = 10000
## tol_rel_grad = 1e+007
## history_size = 5
## seed = 2064940026
## initial log joint probability = -56.7928
## Optimization terminated normally:
##   Convergence detected: relative gradient magnitude is below tolerance
```

```r
    future1 <- make_future_dataframe(m1, periods = 365)
    ##  must specify the caps for this df
    future1$cap <- 12


    forecast1 <- predict(m1, future1)
    plot(m1,forecast1)
```



Trend Changepoints:

Trend Changepoints

- Trend Changepoints are points that specifcy when the rate of the predictions can change.
- By default, prophet calculates many changepoints and uses as little as possible.
- It is possible to manually specify the changepoints via `n.changepoints` parameter in the `prophet` function. To specify the location of changepoints, use `changepoints` parameters.
- To adjust the fitting (over fit or under fit) of trends, the `changepoint.prior.scale` parameter can be changed, default is 0.05. Increasing makes the trend more flexible (overfit potentially) and decreasing it achieves the opposite effect.

Holiday Effect:

- Please consult the holidays documentation. The examples provided helps to make sense of the feature.
- This feature imposes "holiday effects" on the prediction. From the provided example, it seems that it tends to spike up the prediction. **Still needs to confirm that the effect is only increasing prediction.**

Uncertainty Intervals:

- **Uncertainty Interval Documentation**
- **3 major Uncertainties: trend, seasonality and noise:**
- trend:
  - assumes average frequency and magnitude of rate changes are the same as historical data.
  - project these trend changes forward, compute the distribution and the result is an error.
  - changing changepoint prior scale affects the uncertainty because higher scale factor,the more the rate can change at the changepoint.
- noise:
  - demands familiarity with some terms. Please refer to the prophet documentation listed above.

Outliers:

- Outliers
- **It is recommended to look through the examples provided to view the effects.**
- The authors of prophet recommends to remove outliers before prediction. This is because prophet takes into outliers through changing the forecast rate changes. This means outliers effects will be projected into the future permanently.
- Non-Daily Data

Facebook Prophet Articles:

- Facebook Forcasting At Scale
- Taking Prophet for a spin
  -