ChannelAttribution: Markov model for online marketing attribution problem.

Davide Altomare and **David Loris**

www.channelattribution.net

First release: September 1, 2015
Current release: July 7, 2020

ABSTRACT

This paper introduces ChannelAttribution, an open-source library for the estimation of Markov models from customer journey data. ChannelAttribution consists on a R package and a Python library that let to estimate Markov models easily and quickly.

Introduction

Library ChannelAttribution approaches attribution problem in a probabilistic way. It uses a k-order Markov representation to identifying structural correlations in the customer journey data. This would allow advertisers to give a more reliable assessment of the marketing contribution of each channel. The approach is the one presented in F. Anderl, I. Becker, F. v. Wangenheim, J.H. Schumann (2014): Mapping the customer journey: a graph-based framework for attribution modeling. Differently from them, ChannelAttribution uses stochastic simulations for the estimation process. In this way it is also possible to take into account conversion values and their variability in the computation of the channel importance. Moreover the package contains a function that estimates three heuristic models (first-touch, last-touch and linear-touch approach) for the same problem. The following paragraph is a gentle introduction to Markov model. It also contains some considerations on heuristic models.

First-order Markov Model

First-order Markov model is a probabilistic model used to model changing system. It assumes that future states depend only on current state. There is a large leterature about Markov models and different fields where this kind of models have been applied. In the following we will show how they can be applied to attribution problem in online marketing. Consider the following example in which we have 4 states: (START), A, B, (CONVERSION) and 3 paths recorded:

PATH	**CONVERSIONS**
(START) -> A -> B -> A -> B -> B -> A -> (CONV)	1
(START) ->A -> B -> B -> A -> A -> (CONV)	1
(START) -> A -> A -> (CONV)	1
TOTAL	**3**

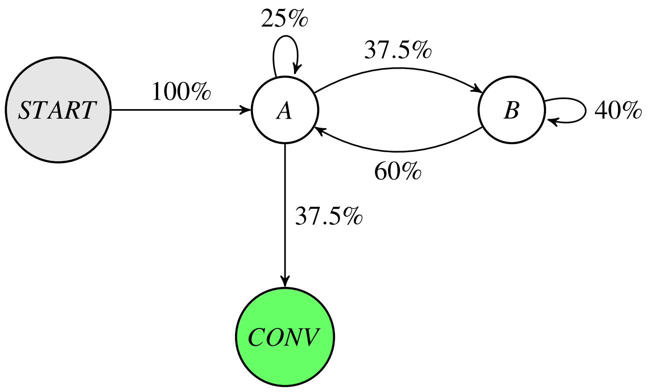
For every couple of ordered states we count the number of directed edges:

EDGE	**ARROW.COUNT**
(START) -> A	3
(START) -> B	0
A -> A	2
A -> B	3
A -> (CONV)	3
B -> A	3
B -> B	2
B -> (CONV)	0
TOTAL	**16**

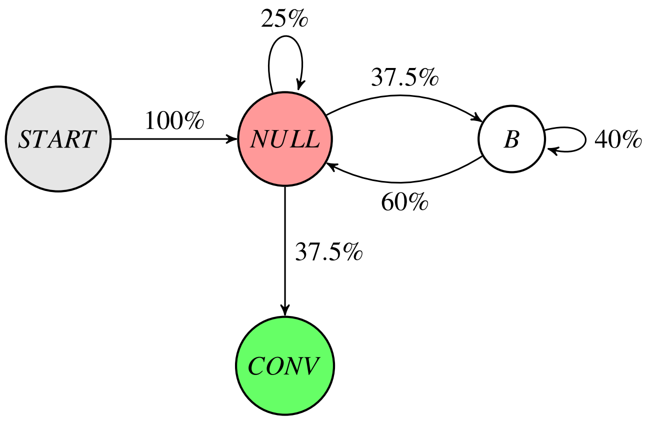
From the table we can calculate the transition probabilities between states:

EDGE	**ARROW.COUNT**	**TRANSITION.PROBABILITIES**
(START) -> A	3	3/3
(START) -> B	0	0/3
TOT (START)	**3**	
A -> A	2	2/8
A -> B	3	3/8
A -> (CONV)	3	3/8
TOT A	**8**	
B -> A	3	3/5
B -> B	2	2/5
B -> (CONV)	0	0/5
TOT B	**5**	

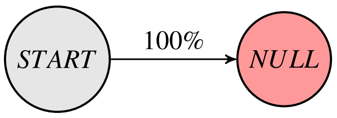
Now we have all the information to plot the Markov Graph:



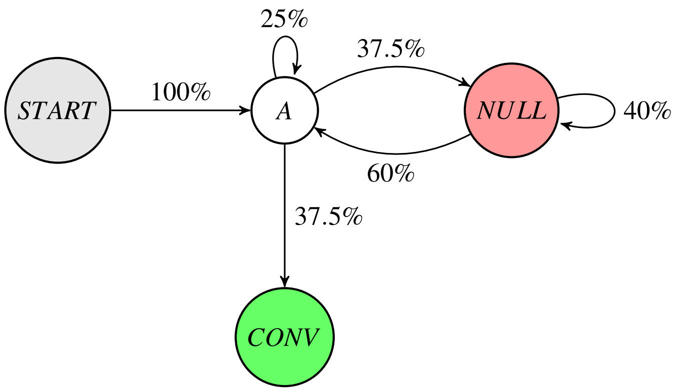
This kind of Markov Graph is called First-Order Markov Graph because the probability of reaching one state depends only on the previous state visited. From the Graph, or more clearly from the original data, we see that every path leads to conversion. Thus the conversion rate of the Graph is 1. Now we want to define a measure of channel importance using the relationship between states described by the Graph. Importance of channel A can be defined as the change in conversion rate if channel A is dropped from the Graph, or in other terms if channel A becomes a NULL state. A NULL state is an absorbing state so if one reaches this STATE can't move on.



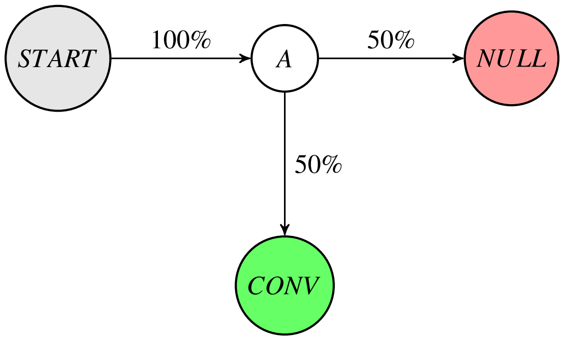
This Graph simplifies to:



In previous Graph it's easy to see that if channel A becomes a NULL state there is no way of reaching conversion from START state. So conversion rate of this Graph is 0. The conversion drops from 1 (conversion of the original Graph) to 0. Thus importance of channel A (defined as the change in conversion rate) is 1. In similar way we define the importance of channel B as the change in conversion rate if channel B is dropped from the Graph, or in other terms if channel B becomes a NULL state.



This Graph simplifies to:



In previous Graph we see the probability of reaching conversion from START state is 0.5. Conversion drops from 1 (conversion rate of the original Graph) to 0.5. Thus the importance of channel B (defined as the change in conversion rate) is **0.5**. Once we have the importance weights of every channel we can do a weighted imputation of total conversions (3 in this case) between channels.

CHANNEL	**CONVERSIONS**
A	2 [=3 x 1/(1+0.5)]
B	1 [= 3 x 0.5/(1+0.5)]
TOTAL	**3**

Now let's go back to the original data:

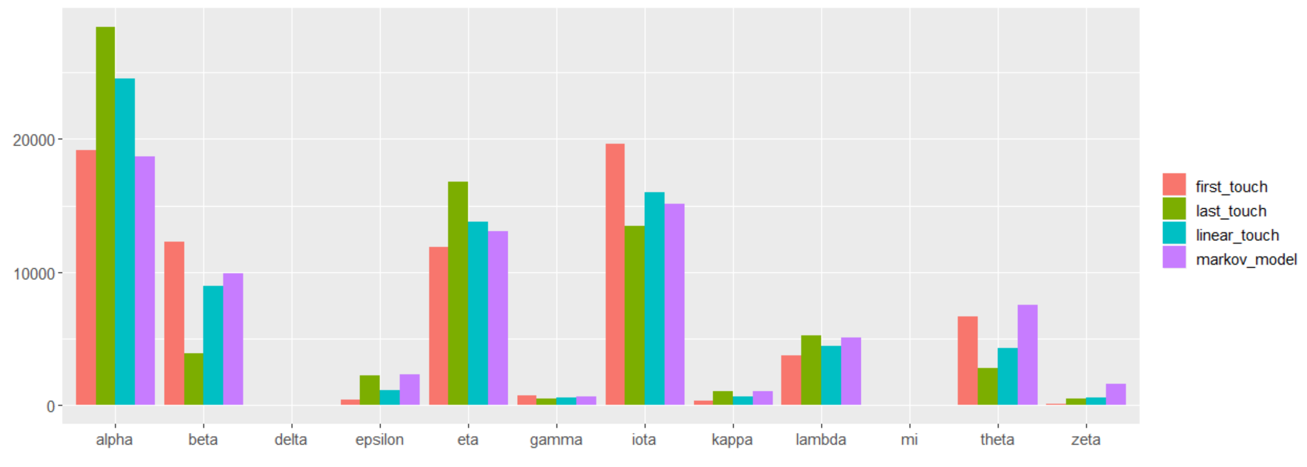
PATH	**CONVERSIONS**
(START) -> A -> B -> A -> B -> B -> A -> (CONV)	1
(START) ->A -> B -> B -> A -> A -> (CONV)	1
(START) -> A -> A -> (CONV)	1
TOTAL	**3**

We see that if we use a first-touch or last-touch approach, all the conversions are assigned to channel A, despite the important work of channel B in "conversion game" is clear from the data. The channel attribution problem can be viewed as a football match, to better understand how different approaches work. So channels can be viewed as players, paths are game actions and conversions are goals. Markov Model analyses relationships between game actions to understand the role of the player in scoring. While heuristic approach analyzes one action (path) at the time. So last-touch approach rewards only players who scored, while first-touch approach rewards only players who started the action. Linear approach rewards with the same credit to every player who took part the action, while time-decay approaches gives subjective weights to every player who took part the action. As we have seen Markov Model require as inputs paths and total conversions and does not require subjective assumptions, differently from heuristic approaches.

R package ChannelAttribution

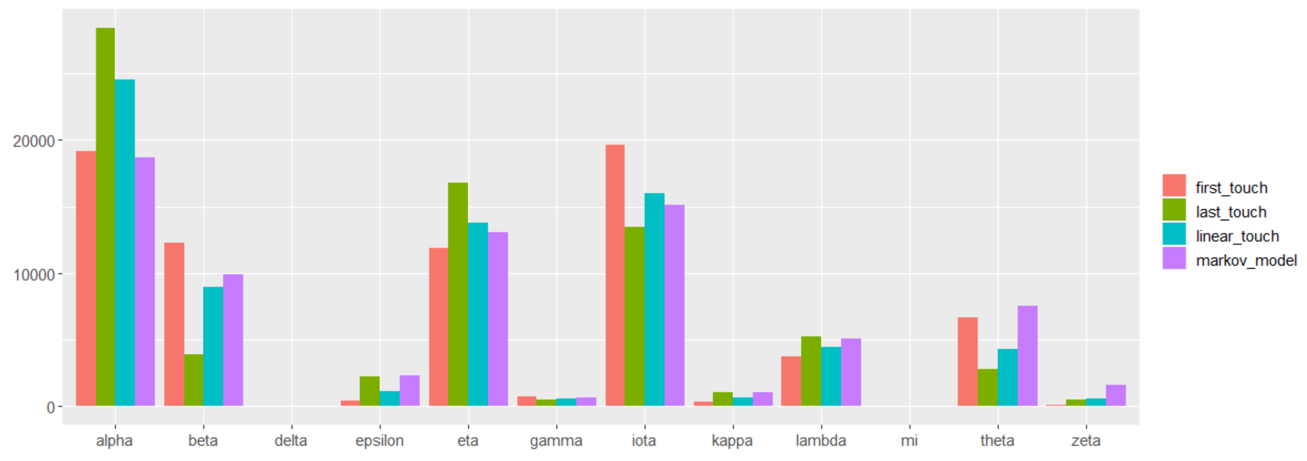
In the following example we will show how R package ChannelAttribution can be used for multichannel attribution problem.

```
#LOAD LIBRARIES AND DATA

library(ChannelAttribution)
library(reshape2)
library(ggplot2)
data(PathData)

#ESTIMATE HEURISTIC MODELS

H=heuristic_models(Data,"path","total_conversions",var_value="total_conversion_value")

#ESTIMATE MARKOV MODEL

M=markov_model(Data, "path", "total_conversions", var_value="total_conversion_value")

#PLOT TOTAL CONVERSIONS

R=merge(H,M,by="channel_name")
R1=R[,(colnames(R)%in%c("channel_name","first_touch_conversions","last_touch_conversions","linear_touch_
    conversions","total_conversion"))]
colnames(R1)=c("channel_name","first_touch","last_touch","linear_touch","markov_model")
R1=melt(R1,id="channel_name")

ggplot(R2, aes(channel_name, value, fill = variable)) +
ggtitle("")+
geom_bar(stat="identity", position = "dodge") +
theme(plot.title = element_text(hjust = 0.5))+
theme(text = element_text(size=14)) +
theme(plot.title=element_text(size=18)) +
theme(legend.title = element_blank()) +
ylab("") +
xlab("")
```



```
#PLOT REVENUES

R2=R[,(colnames(R)%in%c("channel_name","first_touch_value","last_touch_value","linear_touch_value","
    total_conversion_value"))]
```

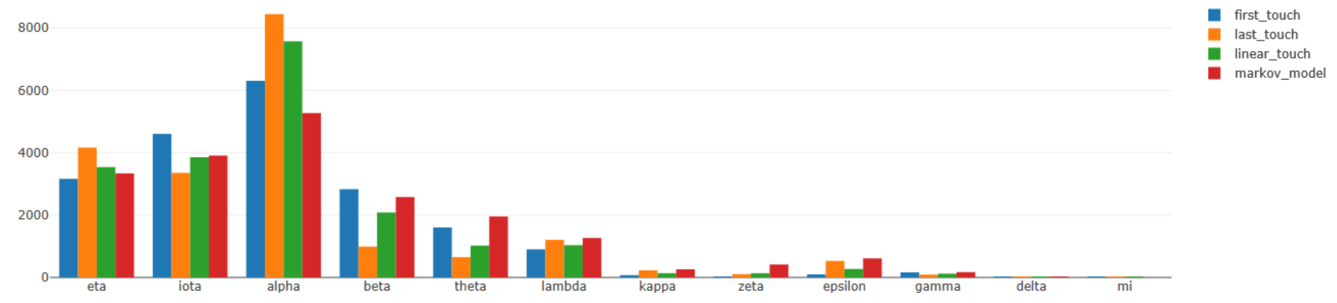
```
colnames(R2)=c("channel_name","first_touch","last_touch","linear_touch","markov_model")
R2=melt(R2,id="channel_name")

ggplot(R2, aes(channel_name, value, fill = variable)) +
ggtitle("")+
geom_bar(stat="identity", position = "dodge") +
theme(plot.title = element_text(hjust = 0.5))+
theme(text = element_text(size=14)) +
theme(plot.title=element_text(size=18)) +
theme(legend.title = element_blank()) +
ylab("") +
xlab("")
```



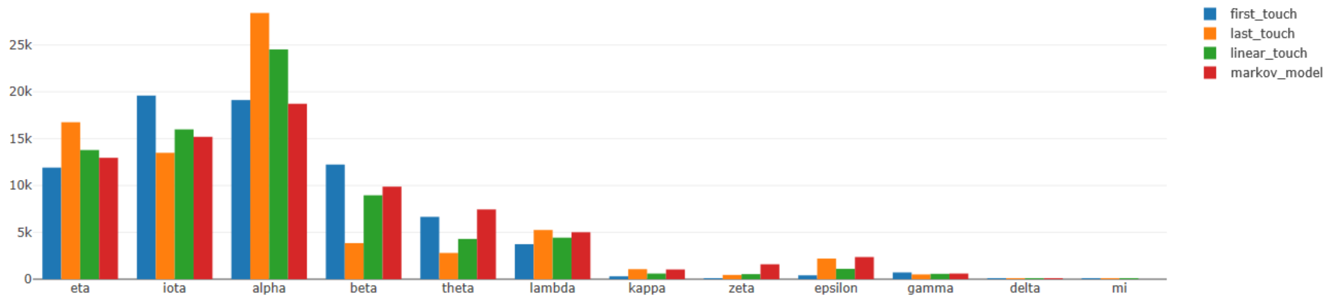
Python library ChannelAttribution

In the following example we will show how Python library ChannelAttribution can be used for multichannel attribution problem.

```
#LOAD LIBRARIES AND DATA

import numpy as np
import pandas as pd
from ChannelAttribution import *
import plotly.io as pio

Data = pd.read_csv("https://raw.githubusercontent.com/DavideAltomare/ChannelAttribution/\
master/ChannelAttribution/src/cypack/data/Data.csv",sep=";")

#ESTIMATE HEURISTIC MODELS

H=heuristic_models(Data,"path","total_conversions",var_value="total_conversion_value")

#ESTIMATE MARKOV MODEL

M=markov_model(Data, "path", "total_conversions", var_value="total_conversion_value")

#PLOT TOTAL CONVERSIONS

R=pd.merge(H,M,on="channel_name",how="inner")
R1=R[["channel_name","first_touch_conversions","last_touch_conversions",\
"linear_touch_conversions","total_conversions"]]
R1.columns=["channel_name","first_touch","last_touch","linear_touch","markov_model"]
R1=pd.melt(R1, id_vars="channel_name")

data = [dict(
  type = "histogram",
  histfunc="sum",
  x = R1.channel_name,
  y = R1.value,
  transforms = [dict(
    type = "groupby",
    groups = R1.variable,
  )],
)]

fig = dict({"data":data})
pio.show(fig,validate=False)
```



```
#PLOT REVENUES

R2=R[["channel_name","first_touch_value","last_touch_value",\
"linear_touch_value","total_conversion_value"]]
R2.columns=["channel_name","first_touch","last_touch","linear_touch","markov_model"]

R2=pd.melt(R2, id_vars="channel_name")

data = [dict(
  type = "histogram",
  histfunc="sum",
  x = R2.channel_name,
  y = R2.value,
  transforms = [dict(
    type = "groupby",
    groups = R2.variable,
  )],
)]

fig = dict(data=data,layout=layout)
\end{comment}

fig = dict({"data":data})
pio.show(fig,validate=False)
```


25k
20k
15k
10k
5k
eta
iota
alpha
beta
theta
lambda
kappa
zeta
epsilon
gamma
delta
mi
first_touch
last_touch
linear_touch
markov_model