

Cross Channel Marketing and Budgeting Optimization:

Criteo has published an online advertising dataset which has details of campaigns and budget allocated for each of them. The aim is to check the cross-channel optimization using various parameters to assess which channel should be getting the maximum budget for advertisement.

Data

The dataset contains the following fields:

- Timestamp: timestamp of the impression
- UID: unique user identifier
- Campaign: unique campaign identifier
- Conversion: 1 if there was a conversion in the 30 days after the impression; 0 otherwise
- Conversion ID: a unique identifier for each conversion
- Click: 1 if the impression was clicked; 0 otherwise
- Cost: the price paid for this ad
- Cat1-Cat9: categorical features associated with the ad. These features' semantic meaning is not disclosed

What is an Attribution Model ?



Attribution modeling is a framework for analyzing which touchpoints/marketing channels receive credit for a conversion.

Each attribution model distributes the credit across each touchpoint differently.

By analyzing each attribution model one can derive the ROI for each marketing channel.

The models that are being used are:

- Last Touch Attribution
- First Touch Attribution
- Time Decay Attribution
- Linear Attribution
- U- Shaped Attribution
- Logistic Attribution

Last Touch Attribution:

The Last Touch Attribution model gives all credits to the last touchpoint in the journey which essentially means that the campaign which is very close to the time a customer converted the ad into revenue.

Last touch helps marketers to prioritize productive channels.

The data is preprocessed to filter them

```
1 def last_touch_attribution(df):
2
3     def count_by_campaign(df):
4         counters = np.zeros(n_campaigns)
5         for campaign_one_hot in df['campaigns'].values:
6             campaign_id = np.argmax(campaign_one_hot)
7             counters[campaign_id] = counters[campaign_id] + 1
8         return counters
9
10    campaign_impressions = count_by_campaign(df)
11
12    df_converted = df[df['conversion'] == 1]
13    idx = df_converted.groupby(['jid'])['timestamp_norm'].transform(max) == df_converted['timestamp_norm']
14    campaign_conversions = count_by_campaign(df_converted[idx])
15
16    return campaign_conversions / campaign_impressions
17
18 lta = last_touch_attribution(df6)
```

First Touch Attribution

The First Touch Attribution Model gives all credits to the first touch point in the marketing journey . 100 % of the credit goes to that particular channel/ activity that acquired or discovered the customer.

First touch attribution emphasizes discovery.

```
def first_touch_attribution(df):

    def count_by_campaign(df):
        counters = np.zeros(n_campaigns)
        for campaign_one_hot in df['campaigns'].values:
            campaign_id = np.argmax(campaign_one_hot)
            counters[campaign_id] = counters[campaign_id] + 1
        return counters

    campaign_impressions = count_by_campaign(df)

    df_converted = df.copy()
    idx = df_converted.groupby(['jid'])['timestamp_norm'].transform(min) == df_converted['timestamp_norm']
    campaign_conversions = count_by_campaign(df_converted[idx])

    return campaign_conversions / campaign_impressions

fta = first_touch_attribution(df6)
```

Linear Attribution Model

The credit for a conversion is equally split between all the campaigns/ touchpoints that a user had in his journey.

Linear attribution gives a more balanced look but fails to highlight the most effective strategies that actually helped in the conversion.

```
def Linear_attribution(df):
    # Assuming the Linear score to be 1 for normalized effect
    linear_score = 1
    def count_by_campaign(df):
        counters = np.zeros(n_campaigns)
        for campaign_one_hot in df['campaigns'].values:
            campaign_id = np.argmax(campaign_one_hot)
            counters[campaign_id] = counters[campaign_id] + 1
        return counters

    def linear_campaign(df):
        counters = np.zeros(n_campaigns)
        for campaign_one_hot in df['campaigns'].values:
            campaign_id = np.argmax(campaign_one_hot)
            for i in range(campaign_id+1):
                counters[i] = (linear_score/campaign_id) # Adding 1 to all campaigns leading up to conversion.
        return counters

    campaign_impressions = count_by_campaign(df)

    dfc = df[df['conversion'] == 1]

    idx = dfc.groupby(['jid'])['timestamp_norm'].transform(max) == dfc['timestamp_norm']
    campaign_conversions = linear_campaign(dfc[idx])

    return campaign_conversions / campaign_impressions

la = Linear_attribution(df6)
```

Time Decay Attribution

Formula - $y = 2^{(-x/7)}$

Time Decay Attribution is similar to linear attribution model and gives importance to all the touchpoints, but it also takes into consideration when the touchpoint occurred.

The interactions that occur closer to the time of conversion get more credit.

```

[ ] def time_decay_attribution(df):

    def count_by_campaign(df):
        counters = np.zeros(n_campaigns)
        for campaign_one_hot in df['campaigns'].values:
            campaign_id = np.argmax(campaign_one_hot)
            counters[campaign_id] = counters[campaign_id] + 1
        return counters

    def discount_by_campaign(df):
        counters = np.zeros(n_campaigns)
        for campaign_one_hot in df['campaigns'].values:
            campaign_id = np.argmax(campaign_one_hot)
            for i in range(campaign_id + 1):

                counters[i] = 2*((campaign_id-i)/campaign_id)
        return counters

    campaign_impressions = count_by_campaign(df)

    dfc = df[df['conversion'] == 1]

    idx = dfc.groupby(['jid'])['timestamp_norm'].transform(max) == dfc['timestamp_norm']
    campaign_conversions = discount_by_campaign(dfc[idx])

    return campaign_conversions / campaign_impressions

tda = time_decay_attribution(df6)

```

U Shaped Attribution

The U shaped attribution model is a multi touch model which counts all the touchpoints with emphasis on the first and the last touchpoints. In the model created, the first and the last touch points have been given a score of 1 and the other points have been given a score 0.5

```
[ ] def U_shape_attribution(df):
    #Assuming u score to be 1 as max and 0.5 as min
    u_score_max = 1
    u_score_min = 0.5
    def count_by_campaign(df):
        counters = np.zeros(n_campaigns)
        for campaign_one_hot in df['campaigns'].values:
            campaign_id = np.argmax(campaign_one_hot)
            counters[campaign_id] = counters[campaign_id] + 1
        return counters

    def ushape_campaign(df):
        counters = np.zeros(n_campaigns)
        for campaign_one_hot in df['campaigns'].values:
            campaign_id = np.argmax(campaign_one_hot)
            for i in range(campaign_id):
                if i == 0:
                    counters[i] += u_score_max
                elif i == campaign_id:
                    counters[i] += u_score_max
                else:
                    counters[i] += u_score_min
            return counters

    campaign_impressions = count_by_campaign(df)

    dfc = df[df['conversion'] == 1]

    idx = dfc.groupby(['jid'])['timestamp_norm'].transform(max) == dfc['timestamp_norm']
    campaign_conversions = ushape_campaign(dfc[idx])

    return campaign_conversions / campaign_impressions

usa = U_shape_attribution(df6)
```

Logistic Regression Attribution:

To identify the real contribution of each touchpoint, logistic regression is used. Here, each journey is represented as a vector in which each campaign is represented by a binary feature, a regression model is fit to predict conversions, and the resulting regression coefficients are interpreted as attribution weights.

```
1 # Quick sanity check
2 from sklearn.linear_model import LogisticRegression
3
4 logisticRegr = LogisticRegression()
5 logisticRegr.fit(x_train, y_train)
6 score = logisticRegr.score(x_test, y_test)
7 print(score)
```

```
1 from keras.models import Sequential
2 from keras.layers import Dense, Dropout
3 from keras.constraints import NonNeg
4
5 m = np.shape(x)[1]
6
7 model = Sequential()
8 model.add(Dense(1, input_dim=m, activation='sigmoid', name = 'contributions'))
9
10 model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
11 history = model.fit(x_train, y_train, batch_size=128, epochs=10, verbose=1, validation_data=(x_val, y_val))
12 score = model.evaluate(x_test, y_test, verbose=0)
13 print('Test score:', score[0])
14 print('Test accuracy:', score[1])
```