ads

December 9, 2019

1 Summary of Findings

1.0.1 Introduction

The following experiment is an attempt to build a regression model that predicts the number of Impressions (views) a Snapchat political advertisement will recieve based on multiple features. These features include the amount spent, the target demographics, and country of origin, for example. The features were selected based on what is currently known about an advertisement before it is published, which is why features such as Ad ID and CreativeURL are not considered. The metric used to verify the strength of the model will be the Root Mean Squared Error, as it allows us to consider the overall performance of our predictions while also punishing single predictions that are extremely far from the actual observed number of Impressions. The R-squared value will also be used as a metric to check how well the model predicts Impression count.

1.0.2 Baseline Model

The Baseline model uses features that are already intuitive examples of predictor variables for an advertisement's Impressions count. This includes the amount spent on an advertisement and the number of days an ad is going to be shown. Both features are quantitative, and no feature engineering was needed to run the regression model. The DaysShown attribute required imputing the null values with the median, as outliers in the dataset skewed the mean significantly. The resulting R-squared value came out to be approximately 0.71, which is a slightly strong suggestion that the predictions are representative of the observed values. The RMSE is relatively large however, approximately 2,260,991 impressions. This is likely due to the presence of multiple outliers and simply because of the low number of features used for prediction initially.

1.0.3 Final Model

The final model utilized a linear regression pipeline with engineered features.

The following nominal attributes were used: 'CountryCode','Gender','OrganizationName','PayingAdvertiserName'.

The following quantitative attributes were used: 'Spend','DaysShown','MinimumAge','MaximumAge'.

The categorical attributes were transformed using a OneHotEncoder, and the numerical attributes were imputed with the mean of their respective column (except for 'Spend', which still used the median). The categorical attributes were chosen because they can be split into discrete groups that may give a better prediction of Impression count depending on the group. For the quantitative

attributes, we still kept Spend and DaysShown from the baseline model, but also included the minimum and maximum age for the ad's target demographic. This is because I hypothesize that having a larger age range may result in an ad recieving more Impressions, since it would encompass a larger number of people who can view the ad. I attempted to include the day of the month and month attributes initially, but those parameters actually increased error overall, thus they were not used in the final model.

Running this more complex regression model resulted in an improved R-squared and RMSE value. The R-squared value came out to approximately 0.77, and the RSME approximately 2,015,447 impressions. The RSME still isn't as good as I would like, but this is still likely due to very large outliers in the dataset.

1.0.4 Fairness Evaluation

In order to better evaluate our final model's fairness, we are going to randomly split the ads dataset into two groups and conduct a permutation test to see if there is a significant difference in R-Squared value, the test statistic being used. We used a p-value of 0.05. The permutation test yielded a p-value of 0.463, which means that there is not a significant difference in r-squared values amongst smaller samples of the dataset. We can also confirm that the model is not overfitted to the ads data we used to build the model initially.

2 Code

[148]:
```
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.decomposition import PCA
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split

%matplotlib inline
%config InlineBackend.figure_format = 'retina'  # Higher resolution figures
```

[149]:
```
# Reading in the two datasets and combining them.
df2018 = pd.read_csv('PoliticalAds2018.csv')
df2019 = pd.read_csv('PoliticalAds2019.csv')

ads = pd.concat([df2018,df2019])
```

[150]:
```
# Data Cleaning: Converting all currency to USD for equal comparison
currencies = ads['Currency Code'].unique()
currencies

ads.loc[ads['Currency Code'] == 'EUR','Spend'] = ads.loc[ads['Currency Code']␣
↪== 'EUR','Spend'] * 1.11
ads.loc[ads['Currency Code'] == 'GBP','Spend'] = ads.loc[ads['Currency Code']␣
↪== 'GBP','Spend'] * 1.29
ads.loc[ads['Currency Code'] == 'CAD','Spend'] = ads.loc[ads['Currency Code']␣
↪== 'CAD','Spend'] * 0.76
ads.loc[ads['Currency Code'] == 'AUD','Spend'] = ads.loc[ads['Currency Code']␣
↪== 'AUD','Spend'] * 0.68

ads.loc[ads['Currency Code'] != 'USD','Currency Code'] = 'USD'
```

[151]:
```
# Data Cleaning: Extracting min and max ages from the AgeBracket attribute
brackets = ads['AgeBracket'].str.replace("+","-").str.replace("--","-").
↪replace(" ","").fillna('').str.split('-')

minAge = brackets.apply(lambda x : x[0])
ads['MinimumAge'] = minAge.replace('',np.nan).astype('float')

def getMaxAge(x):
    if len(x) == 2:
        return x[1]
    else:
        return np.nan
maxAge = brackets.apply(getMaxAge)
ads['MaximumAge'] = maxAge.replace('',np.nan).astype('float')
```

[152]:
```
# Data Cleaning: Converting StartDate and EndDate to datetime objects
ads['StartDate'] = ads['StartDate'].astype('datetime64')
ads['EndDate'] = ads['EndDate'].astype('datetime64')

# Extract the First/Last day of the month, month, and duration of time ad is␣
↪shown
ads['StartDay'] = ads['StartDate'].dt.day
ads['EndDay'] = ads['EndDate'].dt.day
ads['Month'] = ads['StartDate'].dt.month
ads['DaysShown'] = (ads['EndDate'] - ads['StartDate']).dt.days
```

2.0.1 Baseline Model

[153]:
```
# Impute the DaysShown column with the median number of days an ad is shown in␣
↪the dataset.
# The median was chosen as there are a few large outliers that skew the mean.
ads['DaysShown'] = ads['DaysShown'].fillna(ads['DaysShown'].median())
```

[154]:
```
# The baseline model is a simple Linear Regression model with no feature␣
↪engineering.

X = ads[['Spend','DaysShown']]
y = ads['Impressions']

lr = LinearRegression()
lr.fit(X, y)  # X is dataframe of training data; y a series of prices
R_squared = lr.score(X, y)  # R-squared
preds = lr.predict(X) # predicted prices
rmse = np.sqrt(np.mean(np.abs(preds - ads['Impressions'] )**2))
print("r^2: " + str(R_squared) + " RMSE: " + str(rmse))
```

```
r^2: 0.7099881844900044 RMSE: 2260990.7027927483
```

2.0.2 Final Model

[131]:
```
X =␣
↪ads[['CountryCode','Gender','OrganizationName','PayingAdvertiserName','Spend','DaysShown','
y = ads['Impressions']

catcols = ['CountryCode','Gender','OrganizationName','PayingAdvertiserName']
numcols = ['Spend','DaysShown','MinimumAge','MaximumAge']

cats = Pipeline([
    ('imp', SimpleImputer(strategy='constant', fill_value='NULL')),
    ('ohe', OneHotEncoder(handle_unknown='ignore', sparse=False)),
])

ct = ColumnTransformer([
    ('catcols', cats, catcols),
    ('numcols', SimpleImputer(strategy='mean'), numcols)
])

pl = Pipeline([('feats', ct), ('reg', LinearRegression())])
```

[132]:
```
pl.fit(X, y)
R_squared = pl.score(X, y)
preds = pl.predict(X)
```

```
rmse = np.sqrt(np.mean(np.abs(preds - ads['Impressions'] )**2))
print("r^2: " + str(R_squared) + " RMSE: " + str(rmse))
```

```
r^2: 0.7695584173399855 RMSE: 2015446.8751336788
```

2.0.3 Fairness Evaluation

[188]:
```
#Randomly split the dataset in two groups
ads['Group'] = np.nan
ads = ads.assign(Group = np.random.choice(['A','B'],ads['Group'].size,p = [0.
↪5,0.5]))
```

[189]:
```
A = ads.loc[ads['Group'] == 'A']
B = ads.loc[ads['Group'] == 'B']
```

[193]:
```
#Calculate the observed test statistic: The difference in R-Squared values for␣
↪groups A and B
AX =␣
↪A[['CountryCode','Gender','OrganizationName','PayingAdvertiserName','Spend','DaysShown','Mi
Ay = A['Impressions']

pl.fit(AX, Ay)
AR_squared = pl.score(AX, Ay)


BX =␣
↪B[['CountryCode','Gender','OrganizationName','PayingAdvertiserName','Spend','DaysShown','Mi
By = B['Impressions']

pl.fit(BX, By)
BR_squared = pl.score(BX, By)
```

[4]:
```
observed = AR_squared - BR_squared
```

[5]:
```
r2_diffs = []

for i in range(100):
    shuffled = ads.assign(Group=ads.Group.sample(frac=1.0, replace=False).
↪reset_index(drop=True))

    A = shuffled.loc[shuffled['Group'] == 'A']
    B = shuffled.loc[shuffled['Group'] == 'B']

    AX =␣
↪A[['CountryCode','Gender','OrganizationName','PayingAdvertiserName','Spend','DaysShown','Mi
    Ay = A['Impressions']
```

```
    pl.fit(AX, Ay)
    AR_squared = pl.score(AX, Ay)


    BX =␣
↪B[['CountryCode','Gender','OrganizationName','PayingAdvertiserName','Spend','DaysShown','Mi
    By = B['Impressions']

    pl.fit(BX, By)
    BR_squared = pl.score(BX, By)

    r2_diffs.append(AR_squared - BR_squared)
```

[2]:
```
pval = np.mean(r2_diffs > observed)
pval
```

[2]: 0.463