Yichi Zhang Xueru Xie Project 5, DSC 80

Summary

In this project, we decide to investigate the number of views of political ads. Our data come from datasets ads in the year 018 and 2019 on Snapchat, where the column 'Impressions' denotes the total number of views of the ad. We will be using these data to fit a linear regression model and try to predict the number of views of an ad by its spending, geographic location, as well as the length of time(in hours) that the ad is propagated.

We construct our baseline model based on the linear regression. The figure of Spend-Impression distribution below (in the code) shall justify our choice by its shape.

For our baseline model, we include features such as the Spend and Regions(encoded from nominal values) columns from the given data, and reach a score around -6, which means our baseline model is not performing well(in fact, worse than mean-predictor). Such a result to some degree is determined, since we only include 2 variables and one of which has 3/4 of its inputs being missing. This means our model is better performed without regions variables. Also, from a perspective of maintaining comprehensiveness, the region variable for it only applies to 1/4 (about 1000+) of data. Combining the above 2 reason, we decide to give up the region values and try just using Spend-Impression to construct a simple linear model. Our R^2 score after the alternation is around 0.39, which is greatly improved.

Next, we add more engineered features in. These features include the length of time the add has been spread and if the ad is considerably expensive as compared to other ads. We use these as features for good reasons. According to our research in proj3, cheap ads and much expensive ads perform differently and shows different properties. To avoid bias on one of the kinds, we introduce a binomial variable "ExpensiveAds" that, as we did in proj3, categorize top 25% costly ads as expensive and the rest as non-expensive. The second feature we introduce is the length of spreading time of an ad, we choose this since by common sense ads that exists for longer usually attracts more viewers.

For both above features, we fail to include their generic form in our baseline model due to the lack of appropriate ways of doing so. The typical reason is that the time variable, calculated by EndDate-StartDate, does not have a generic form, using either End or Start date would be inappropriate. The same goes to the "expensiveness", since we've already included spend in our model.

For the two engineered features, we split the data into training and test data. Then, we use linear regression model to predict and score it. The score is the number that evaluates the accuracy(R-square) of our model. Our score is around 0.39, and we believe that it is bad because it does not accurately predict the Impression column. So, we want to add more features to improve our model.

For our final model, we use the Spend, Time, and Expensiveness for the features. We run the linear regression model again on the test dataset. This time, our score is around 0.66. This score is higher than the baseline model, which means that our final model is improved.

In [211]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
```

In [267]:
```
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import Binarizer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
```

In [213]:
```
df = pd.read_csv('PoliticalAds2018.csv')
df1 = pd.read_csv('PoliticalAds2019.csv')
```

In [214]:
```
# data joining and cleaning
snap = pd.concat([df,df1],axis = 0)
snap['StartDate']  = pd.to_datetime(snap['StartDate'])
snap['EndDate']  = pd.to_datetime(snap['EndDate'])
snap.head()
def currency(money):
    if money[0] == 'USD':
        return money
    elif money[0] == 'GBP':
        money[0] = 'USD'
        money[1] *= 1.22
    elif money[0] == 'EUR':
        money[0] = 'USD'
        money[1] *= 1.08
    elif money[0] == 'CAD':
        money[0] = 'USD'
        money[1] *= 0.71
    elif money[0] == 'AUD':
        money[0] = 'USD'
        money[1] *= 0.65
    return money

new = []
for i in snap[['Currency Code', 'Spend']].values:
    new.append(currency(i))
snap[['Currency Code', 'Spend']] = new
snap.head()
```

Out[214]:

	ADID	CreativeUrl	Currency Code	Spen
0	093af971d3e11037529c7e6815ccd14b5c6336e19d8ff4...	https://www.snap.com/political-ads/asset/586a7...	USD	1249.
1	ccdfa07793d238a537f74999c94406aeca9b847c1ca20f...	https://www.snap.com/political-ads/asset/51f99...	USD	138.
2	bab8a42a290cfd2ac6bdf43c58db4b03d97155565a09cc...	https://www.snap.com/political-ads/asset/0116b...	USD	1000.
3	fc698d5796ce58b5bd40a887fa04aa9e06b1db032e57e7...	https://www.snap.com/political-ads/asset/b1583...	USD	156.
4	9ca1da9db1c9d7b39e08ce075a81bd2bb0a8c2ccecd4fc...	https://www.snap.com/political-ads/asset/6bfcd...	USD	9.

5 rows × 34 columns

In [ ]:

```
```

Baseline Model

In [215]:

```
# construct dataset for modeling
ads = snap[['Impressions','Spend']] # data for baseline model
```

In [216]:

```
from sklearn.preprocessing import LabelEncoder
```

In [217]:

```
region_df=pd.DataFrame(snap['Regions (Included)'])
mask = region_df.isnull()
original = region_df
```

In [218]:

```
region_df = region_df.astype(str).apply(LabelEncoder().fit_transform)
region_endoced = region_df.where(~mask, original)
```

In [219]:

```
ads['Regions'] = region_endoced['Regions (Included)'] # add feature 0
```

```
E:\ANACONDA\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  """Entry point for launching an IPython kernel.
```

In [220]:

```
time = snap.StartDate - snap.EndDate
time.values.astype('timedelta64[h]')
```

Out[220]:

```
array([ -637, 'NaT',  -168, ...,  -174, -1668,  -433],
      dtype='timedelta64[h]')
```

In [221]:

```
time = snap.StartDate - snap.EndDate
ads['Time'] = time.values.astype('timedelta64[h]')/ np.timedelta64(1, 'h') # add
```

```
E:\ANACONDA\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
```

In [222]:
```
threshold = np.quantile(np.unique(snap['Spend']),0.75)
ads['ExpensiveAds'] = snap['Spend'] > threshold # add feature 2, nominal/binomial
```

```
E:\ANACONDA\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
```

In [223]:
```
ads.head()
```

Out[223]:

	Impressions	Spend	Regions	Time	ExpensiveAds
0	274231	1249.0	NaN	-637.0	False
1	77778	138.0	NaN	NaN	False
2	274592	1000.0	NaN	-168.0	False
3	57789	156.0	NaN	-99.0	False
4	1521	9.0	NaN	-174.0	False

In [224]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

In [225]:
```
base = ads.dropna()
```

In [226]:
```
base.head()
```

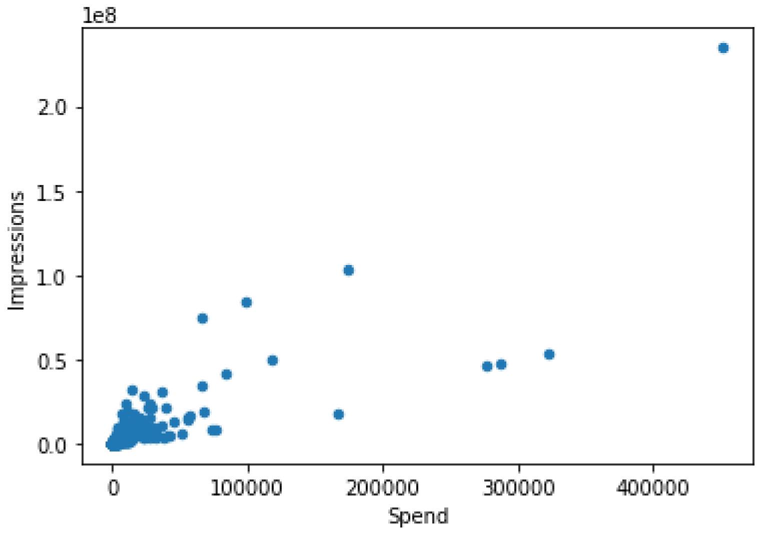
Out[226]:

	Impressions	Spend	Regions	Time	ExpensiveAds
6	691458	1706.0	148	-684.0	True
7	2280127	8547.0	68	-131.0	True
13	229754	775.0	67	-126.0	False
14	3373594	11977.0	71	-268.0	True
17	19330	134.0	56	-463.0	False

Use linear model for spend-impression relationship

```
In [261]: ads[['Spend','Impressions']].plot.scatter(x = 'Spend', y = 'Impressions') # distr

Out[261]: <matplotlib.axes._subplots.AxesSubplot at 0x1b881385cf8>
```



```
In [227]: X = base[['Spend','Regions']]
          y = base.Impressions.values

In [317]: X_test, y_train, y_test = train_test_split(X, y, random_state=0, test_size = 0.3)

In [318]: pl = Pipeline([('lr', LinearRegression())])
          pl.fit(X_train, y_train)

Out[318]: Pipeline(memory=None,
                   steps=[('lr',
                           LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                                            normalize=False))],
                   verbose=False)
```

In [319]:
```
pl.predict(X)
```
Out[319]:
```
array([ 7.00671048e+05,  3.79565401e+06,  3.31310481e+05,  5.32312176e+06,
        5.17460970e+04,  2.13775485e+05,  4.97438945e+05,  2.07491057e+05,
        2.70295088e+06, -8.08767453e+03,  1.31583430e+05,  9.41755463e+05,
        1.64356807e+05, -6.23304533e+03,  3.44200370e+05,  9.45221191e+04,
        9.30815454e+02,  6.42654696e+03, -2.22436687e+04,  4.16426294e+05,
       -2.84770191e+04,  8.89322240e+05, -6.21360942e+03, -1.05810147e+04,
        3.12205237e+04,  9.79014048e+06, -8.41515003e+03,  1.19608160e+06,
        3.95939770e+05,  2.27445898e+05, -1.73396563e+04,  5.44002852e+06,
        1.59540882e+04,  4.05002056e+05,  6.46578832e+04,  1.80543886e+06,
        7.76645102e+05,  1.87554679e+04,  1.17454992e+05,  7.37487550e+05,
       -1.77485222e+03, -4.90430152e+04,  7.33309841e+05,  2.14915965e+05,
        3.17004680e+04,  1.55853270e+06,  1.38716539e+05,  1.00137691e+06,
        3.43591438e+04,  4.88896054e+04,  8.82174168e+04,  2.21842427e+04,
        6.26151066e+05,  2.68779302e+06,  4.07487276e+05,  7.21273215e+04,
        3.81737018e+04,  4.13472623e+05, -2.13714660e+04, -3.96808096e+04,
        5.71125595e+05,  1.07640575e+05,  2.95025661e+05,  1.20517419e+05,
        4.36815639e+05,  4.59164983e+04,  9.51283004e+05,  1.52228919e+05,
        6.02771951e+03,  1.50624496e+04,  4.84633453e+05,  1.63821928e+05,
       -5.32197080e+03, -1.13933714e+03,  6.29981864e+05,  1.05490044e+07,
```

In [320]:
```
pl.score(X_test, y_test) #baseline model 1
```
Out[320]:
```
-5.896276562135982
```

In [321]:
```
test_result = pl.predict(X_test)
```

In [322]:
```
np.sqrt(np.mean(test_result - y_test)**2)
```
Out[322]:
```
263669.0356409888
```

In [ ]:
```

```

In [323]:
```
X1 = ads[['Spend']]
y1 = ads.Impressions.values
```

In [324]:
```
t1, y_train1, y_test1 = train_test_split(X1, y1, random_state=0, test_size = 0.3)
```

In [325]:
```
pl1 = Pipeline([('lr1', LinearRegression())])
pl1.fit(X_train1, y_train1)
```
Out[325]:
```
Pipeline(memory=None,
         steps=[('lr1',
                 LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                                  normalize=False))],
         verbose=False)
```

In [326]:
```
pl1.predict(X1)
```
Out[326]:
```
array([587336.26100142, 191364.17158344, 498590.04114087, ...,
       157505.17203825, 461523.34690192, 152515.42473686])
```

In [327]:
```
pl1.score(X_test1, y_test1) #baseline model 2 (improved)
```
Out[327]:
```
0.39564142266745617
```

In [328]:
```
preds = pl1.predict(X_test1)
```

Final Model

Features 'Time' and 'ExpensiveAds' already engineered before, generic feature of both does not apply becuase of their properties

In [329]:
```
new = ads.drop(columns=['Regions'])
new = noregion.dropna()
new.head()
```
Out[329]:

	Impressions	Spend	Time	ExpensiveAds
0	274231	1249.0	-637.0	False
2	274592	1000.0	-168.0	False
3	57789	156.0	-99.0	False
4	1521	9.0	-174.0	False
5	53211	277.0	-493.0	False

In [330]:
```
def tf(s):
    if s == False:
        return 0
    if s == True:
        return 1
```

In [331]:
```
new['ExpensiveAds'] = new.ExpensiveAds.apply(tf)
```

In [332]:
```
new.head()
```
Out[332]:

	Impressions	Spend	Time	ExpensiveAds
0	274231	1249.0	-637.0	0
2	274592	1000.0	-168.0	0
3	57789	156.0	-99.0	0
4	1521	9.0	-174.0	0
5	53211	277.0	-493.0	0

In [333]:
```
A = new.drop(['Impressions'], axis = 1)
b = new.Impressions.values
```

In [334]:
```
pl2 = Pipeline(steps=[('regressor', LinearRegression())])
```

In [335]:
```
est2, y_train2, y_test2 = train_test_split(A, b, random_state=0, test_size = 0.3)
```

In [336]:
```
pl2.fit(X_train2, y_train2)
```

Out[336]:
```
Pipeline(memory=None,
         steps=[('regressor',
                 LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                                  normalize=False))],
         verbose=False)
```

In [337]:
```
pl2.score(X_test2, y_test2) #final model
```

Out[337]:
```
0.6697336809502963
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: