ads

March 15, 2020

1 Snapchat Political Ads

- **See the main project notebook for instructions to be sure you satisfy the rubric!**
- See Project 03 for information on the dataset.
- A few example prediction questions to pursue are listed below. However, don't limit yourself to them!
  - Predict the reach (number of views) of an ad.
  - Predict how much was spent on an ad.
  - Predict the target group of an ad. (For example, predict the target gender.)
  - Predict the (type of) organization/advertiser behind an ad.

Be careful to justify what information you would know at the "time of prediction" and train your model using only those features.

2 Summary of Findings

2.0.1 Introduction

This project will focus on predicting the country of origin of an ad. This will be a classification problem since the goal is to predict a categorical variable, rather than a continuous variable. In order to predict the country of origin, this project will use CountryCode as the target variable. The objective will be to maximize the accuracy of the model. Given the malicious ad campaigns that have taken place in the US by foreign governments, this project asks whether it is feasible to know where an ad originated from, with only knowing some basic information of the ad. For the purpose of addressing this question we will exclude billing address from the data as well.

2.0.2 Baseline Model

For a baseline model I decided to first clean up my data a little. I first filled null values with their appropriate meanings based on snapchat's readme file for the data. As well, I excluded columns that had over 95% null values as I do not think they would provide a better prediction. I ended up using 16 feature variables. Of the 16, 14 of the features were nominal and 2 features were quantitative. As a baseline model, I decided to one hot encode all of the features except the two quantitative features. I used a Random Forest Classifier and was able to achieve an average accuracy of approximately 63%. I used cross validation as well as an accuracy score based on a held out test sample of the data to make sure my accuracy was consistent. For a baseline model, I feel this is a decent outcome since it is better than just randomly guessing the country of origin.

Also, it is decently high for just one hot encoding the original features, so I believe there is a lot of room for improvement.

2.0.3 Final Model

I ended up adding two new features to my data. I first chose to create a column called Days which calculates the time the ad was shown on snapchat by subtracting the start date from the end date. I think this column might be helpful in predictions in case there are associations with the length of time an ad is shown and the country the ad is from. Also, I wanted to make better use of my start date and end date data, rather than just one hot encoding them.

I also changed the age bracket column to include only 5 different groups. I did this because originally the age bracket column contained all sorts of values and in different formats. I thought this column might have an association with the country of origin so I used regex to extract the age groups and bin the ages into fewer categories. This also works better for using one hot encoding.

I ended up choosing a random forest classifier model for my data. I also tried a k nearest neighbor approach, but the results were not nearly as good. I ran grid search on my random forest model and ended up finding the best features to be: 'criterion': 'gini','max_depth': 10,'max_features': 'sqrt','min_samples_leaf': 1, 'min_samples_split': 3,'n_estimators': 100. My method of model selection included evaluating two different classifiers, random forest and knn, by first building a pipeline for the transformations and construction of the model then performing grid search on both and calculating accuracy scores based on cross validation as well as a held out test sample. What I found was that the random forest classifier outperformed the knn classifier by over 20%. In the end I was able to improve my baseline model by slightly over 10% up to around 74% accuracy.

2.0.4 Fairness Evaluation

To evaluate fairness I split my dataset into two subsets. The data is split by impressions with one subset being the ads with number of impressions that are below the median number of impressions for all ads. The other subset is ads with number of impressions that are equal to or greater than the median number of impressions for all ads. Since my target variable is categorical, it does not make sense to choose a parity measure, so I used accuracy. I then performed a permutation test between my two subsets and used the difference of means for my test statistic. My hypotheses are outlined below:

Null Hypothesis: The model predicts with the same accuracy for ads with high number of impressions as ads with low number of impressions.

Alternative Hypothesis: The model predicts with a different accuracy for ads with a high number of impressions as ads with low number of impressions.

Note: We define high number of impressions to be ads with impressions equal to or greater than the median impressions for all ads, and low number of impressions is ads with impressions less than the median.

Decision: We will set a significance level of .05. So we will reject the null hypothesis for a p value less than .05 and fail to reject the null hypothesis if it is greater than .05.

After running my hypothesis test I got a p value of .61 so I fail to reject my null hypothesis. Thus, between ads with high number of impressions and low number of impressions my model predicts both groups fairly equally.

3 Code

```
In [1]: import matplotlib.pyplot as plt
        import numpy as np
        import os
        import pandas as pd
        import seaborn as sns
        %matplotlib inline
        %config InlineBackend.figure_format = 'retina'  # Higher resolution figures
```

```
In [2]: from sklearn.linear_model import LinearRegression
        from sklearn.preprocessing import FunctionTransformer
        from sklearn.preprocessing import OneHotEncoder
        from sklearn.preprocessing import LabelEncoder
        from sklearn.pipeline import Pipeline
        from sklearn.compose import ColumnTransformer

        from sklearn.tree import DecisionTreeRegressor
        from sklearn.model_selection import train_test_split
        from sklearn.neighbors import KNeighborsRegressor
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import mean_squared_error
        from sklearn.decomposition import PCA
        from sklearn.svm import SVC
        from sklearn.model_selection import cross_val_score
        from sklearn.metrics import make_scorer, accuracy_score
        from sklearn.model_selection import GridSearchCV
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.neighbors import KNeighborsClassifier

        from datetime import datetime
        import re
```

3.0.1 Baseline Model

```
In [14]: # Importing the data
         ads_fp = os.path.join('data', 'PoliticalAds2018.csv')
         ads2018 = pd.read_csv(ads_fp)
         ads_fp = os.path.join('data', 'PoliticalAds2019.csv')
         ads2019 = pd.read_csv(ads_fp)
         ads = ads2018.append(ads2019)
         orig_ads = ads.copy()
```

First some basic cleaning

```
In [15]: # Replacing null values with actual value the null is representing
         ads['Gender'] = ads['Gender'].fillna('ALL')
         ads['AgeBracket'] = ads['AgeBracket'].fillna('-1')
         ads['Interests'] = ads['Interests'].fillna('NONE')
```

```
ads['Language'] = ads['Language'].fillna('NONE')
ads['CandidateBallotInformation'] = (ads['CandidateBallotInformation'].
                                     fillna('NONE'))
ads['Regions (Included)'] = ads['Regions (Included)'].fillna('NONE')
ads['Radius Targeting (Included)'] = (ads['Radius Targeting (Included)'].
                                      fillna('NONE'))
ads['Postal Codes (Included)'] = (ads['Postal Codes (Included)'].
                                  fillna('NONE'))
ads['EndDate'] = ads['EndDate'].fillna('ONGOING')
ads['Segments'] = ads['Segments'].fillna('NONE')
```

In [16]:
```
# Dropping columns that have over 95 % null values
cols = ['Regions (Excluded)', 'Electoral Districts (Excluded)',
        'Electoral Districts (Included)', 'Radius Targeting (Excluded)',
        'Metros (Excluded)', 'Metros (Included)', 'Postal Codes (Excluded)',
        'Location Categories (Excluded)', 'Location Categories (Included)',
        'OsType', 'AdvancedDemographics', 'Targeting Connection Type',
        'Targeting Carrier (ISP)', 'CreativeUrl','CreativeProperties']
ads = ads.drop(cols, axis=1)
# Drop ADID since it is a unique value for each ad
ads = ads.drop(['ADID'], axis=1)
# Dropping billing address since the point of the model is to predict
# origin location
ads = ads.drop('BillingAddress', axis=1)
```

In [17]:
```
# Choosing baseline features to use and creating a transformer to hold
# those features
transformer = ColumnTransformer([
    ('one_hot_curr', OneHotEncoder(), ['Currency Code']),
    ('one_hot_start', OneHotEncoder(handle_unknown='ignore'),
     ['StartDate']),
    ('one_hot_end', OneHotEncoder(handle_unknown='ignore'),
     ['EndDate']),
    ('one_hot_org', OneHotEncoder(handle_unknown='ignore'),
     ['OrganizationName']),
    ('one_hot_cand', OneHotEncoder(handle_unknown='ignore'),
     ['CandidateBallotInformation']),
    ('one_hot_pay', OneHotEncoder(handle_unknown='ignore'),
     ['PayingAdvertiserName']),
    ('one_hot_gender', OneHotEncoder(), ['Gender']),
    ('one_hot_age', OneHotEncoder(handle_unknown='ignore'),
     ['AgeBracket']),
    ('one_hot_regions', OneHotEncoder(handle_unknown='ignore'),
     ['Regions (Included)']),
    ('one_hot_radius', OneHotEncoder(handle_unknown='ignore'),
     ['Radius Targeting (Included)']),
    ('one_hot_postal', OneHotEncoder(handle_unknown='ignore'),
     ['Postal Codes (Included)']),
```

```
            ('one_hot_int', OneHotEncoder(handle_unknown='ignore'),
             ['Interests']),
            ('one_hot_seg', OneHotEncoder(handle_unknown='ignore'),
             ['Segments']),
            ('one_hot_lan', OneHotEncoder(handle_unknown='ignore'),
             ['Language'])
        ], remainder='passthrough')
```

In [19]:
```
import warnings
warnings.filterwarnings('ignore')
# Creating a pipeline to process transformations and execute random
# forest tree
pl = Pipeline([
        ('transformations', transformer),
        ('rf', RandomForestClassifier(bootstrap=True,
                class_weight=None, criterion='gini',
                max_depth=5, max_features='auto',
                max_leaf_nodes=None, min_samples_leaf=1,
                min_samples_split=2, min_weight_fraction_leaf=0.0,
                n_estimators=100, n_jobs=1, oob_score=False,
                random_state=None, verbose=0,warm_start=False))
])
scores = dict()
# Getting the X and y data
X = ads.drop('CountryCode', axis=1)
y = ads['CountryCode']
# Getting the training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
# Fitting the data
pl.fit(X_train, y_train)
# Getting the accuracy score
scores['test_accuracy'] = (accuracy_score(y_test, pl.predict(X_test)))
scores['validation_accuracy'] = np.mean(cross_val_score(
    pl, X_train, y_train, cv=4))
scores
```

Out[19]:
```
{'test_accuracy': 0.6049960967993755,
 'validation_accuracy': 0.6370903632402711}
```

First let's add some features

In [20]:
```
# Replacing 'EndDate' null values with the current date to represent
# they are still running the ad
replace = str(datetime.now())
ads['EndDate'] = orig_ads['EndDate'].fillna(replace)
# Using 'StartDate' and EndDate to create a column 'Days' that denotes
# the number of days the ad has been showing
duration = ((pd.to_datetime(ads['EndDate']) - pd.to_datetime(
```

```
            ads['StartDate']))).dt.days
        ads['Days'] = duration
```

In [21]:
```
def bin_ages(nums):
    if int(nums[0]) == -1:
        return 'all'
    elif int(nums[0]) >= 18 & len(nums) > 1:
        if int(nums[1]) < 40:
            return 'young_adult'
        if int(nums[1]) >= 40:
            return 'adult'
    elif int(nums[0]) >= 18 & len(nums) == 1:
        return 'adult'
    elif int(nums[0]) < 18 & len(nums) > 1:
        if int(nums[1]) < 18:
            return 'child'
        if int(nums[1]) > 18 & int(nums[1]) < 40:
            return 'child_young_adult'
        if int(nums[1]) > 18 & int(nums[1]) >= 40:
            return 'all'
    elif int(nums[0]) < 18 & len(nums) == 1:
        return 'child'
    else:
        return 'adult'
```

In [22]:
```
# Let's clean up and categorize the age brackets into child (<18),
# young adult (>18 and < 40), adult (>18 and >40), child young
# adult (<18 and <40), and all
brackets = ads['AgeBracket'].apply(lambda x: re.findall(r'\d+', x))
brackets = brackets.apply(bin_ages)
ads['AgeBracket'] = brackets
```

In [23]:
```
# Using the same transformer as above, but now we have the two
# added features to build a pipeline
# Creating a pipeline to process transformations and execute
# random forest tree
pl = Pipeline([
        ('transformations', transformer),
        ('rf', RandomForestClassifier(bootstrap=True,
                class_weight=None, criterion='gini',
                max_depth=5, max_features='auto',
                max_leaf_nodes=None, min_samples_leaf=1,
                min_samples_split=2, min_weight_fraction_leaf=0.0,
                n_estimators=100, n_jobs=1, oob_score=False,
                random_state=None, verbose=0,warm_start=False))
])
# Getting the X and y data along with training and testing data
X = ads.drop('CountryCode', axis=1)
```

```
        y = ads['CountryCode']
        X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                            test_size=0.3)
        # Fitting the data
        pl.fit(X_train, y_train)
        # Getting the accuracy score
        accuracy_score(y_test, pl.predict(X_test))
```

```
Out[23]: 0.6479313036690086
```

```
In [134]: # Let's run grid search to find the best model fit
          parameters1 = {'rf__n_estimators': [95, 100],
                         'rf__max_features': ['log2', 'sqrt','auto'],
                         'rf__criterion': ['entropy', 'gini'],
                         'rf__max_depth': [3, 5, 7, 10],
                         'rf__min_samples_split': [2, 3, 5],
                         'rf__min_samples_leaf': [1,5,8]
                        }
          grid = GridSearchCV(pl, parameters1, cv=3)
          grid = grid.fit(X_train, y_train)
```

```
In [135]: # Best parameters
          grid.best_params_
```

```
Out[135]: {'rf__criterion': 'gini',
           'rf__max_depth': 10,
           'rf__max_features': 'auto',
           'rf__min_samples_leaf': 1,
           'rf__min_samples_split': 3,
           'rf__n_estimators': 100}
```

```
In [24]: # Now lets take the best parameters and use them in our model
         pl = Pipeline([
                 ('transformations', transformer),
                 ('rf', RandomForestClassifier(bootstrap=True,
                         class_weight=None, criterion='gini',
                         max_depth=10, max_features='sqrt',
                         max_leaf_nodes=None, min_samples_leaf=1,
                         min_samples_split=3, min_weight_fraction_leaf=0.0,
                         n_estimators=100, n_jobs=1, oob_score=False,
                         random_state=None, verbose=0,warm_start=False))
         ])
         scores = dict()
         # Getting the X and y data
         X = ads.drop('CountryCode', axis=1)
         y = ads['CountryCode']
         # Getting the training and testing data
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
         # Fitting the data
```

```
pl.fit(X_train, y_train)
# Getting the accuracy score
scores['test_accuracy'] = (accuracy_score(y_test, pl.predict(X_test)))
scores['validation_accuracy'] = np.mean(cross_val_score(
    pl, X_train, y_train, cv=4))
scores
```

```
Out[24]: {'test_accuracy': 0.7439500390320063, 'validation_accuracy': 0.757960348992036}
```

Impressively, just by adding our two new features and running Grid Search cross validation we are able to consistently improve our model by over 10 percentage points in accuracy.

Let's try a different classifier now

```
In [202]: # Trying knn
          pl = Pipeline([
                  ('transformations', transformer),
                  ('knn', KNeighborsClassifier())
          ])
          scores = dict()
          # Getting the X and y data
          X = ads.drop('CountryCode', axis=1)
          y = ads['CountryCode']
          # Getting the training and testing data
          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
          # Fitting the data
          pl.fit(X_train, y_train)
          # Getting the accuracy score
          scores['test_accuracy'] = (accuracy_score(y_test, pl.predict(X_test)))
          scores['validation_accuracy'] = np.mean(cross_val_score(
              pl, X_train, y_train, cv=4))
          scores
```

```
Out[202]: {'test_accuracy': 0.419984387197502, 'validation_accuracy': 0.4385737767872204}
```

```
In [138]: # Let's use grid search to see if we can improve it
          parameters2 = {'knn__n_neighbors':[2,3,5,7,9,11],
                         'knn__weights':['uniform', 'distance'],
                         'knn__algorithm':['auto', 'ball_tree', 'kd_tree', 'brute'],
                         'knn__leaf_size':[25, 30, 35],}
          grid = GridSearchCV(pl, parameters2, cv=3)
          grid = grid.fit(X_train, y_train)
```

```
In [139]: grid.best_params_
```

```
Out[139]: {'knn__algorithm': 'auto',
           'knn__leaf_size': 25,
           'knn__n_neighbors': 11,
           'knn__weights': 'uniform'}
```

```
In [239]: # Using the found parameters
          pl = Pipeline([
                  ('transformations', transformer),
                  ('rf', KNeighborsClassifier(algorithm='auto',
                                              leaf_size=25,
                                              n_neighbors=11,
                                              weights='uniform'))
          ])
          scores = dict()
          # Getting the X and y data
          X = ads.drop('CountryCode', axis=1)
          y = ads['CountryCode']
          # Getting the training and testing data
          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
          # Fitting the data
          pl.fit(X_train, y_train)
          # Getting the accuracy score
          scores['test_accuracy'] = (accuracy_score(y_test, pl.predict(X_test)))
          scores['validation_accuracy'] = np.mean(cross_val_score(
              pl, X_train, y_train, cv=4))
          scores

Out[239]: {'test_accuracy': 0.4769711163153786,
           'validation_accuracy': 0.48912405654790747}
```

Not able to get very good results using knn

3.0.2 Fairness Evaluation

Now that we have our model let's evaluate the performance on two subsets of the data.

```
In [25]: # Using the median to split the ads into two subsets
         med = ads['Impressions'].median()
         low_ads = ads[ads['Impressions'] < med]
         high_ads = ads[ads['Impressions'] >= med]
         X = ads.drop('CountryCode', axis=1)
         y = ads['CountryCode']
```

```
In [28]: # Now lets find the accuracy of low and high by retraining the model
         scores = dict()
         pl = Pipeline([
                 ('transformations', transformer),
                 ('rf', RandomForestClassifier(bootstrap=True,
                         class_weight=None, criterion='gini',
                         max_depth=10, max_features='sqrt',
                         max_leaf_nodes=None, min_samples_leaf=1,
                         min_samples_split=3, min_weight_fraction_leaf=0.0,
                         n_estimators=100, n_jobs=1, oob_score=False,
                         random_state=None, verbose=0,warm_start=False))
```

```
])
pl.fit(X, y)
# Getting the X and y data
X = low_ads.drop('CountryCode', axis=1)
y = low_ads['CountryCode']

# Getting the accuracy score
accuracy = (accuracy_score(y, pl.predict(X)))
scores['low'] = accuracy

# Now find accuracy of high
# Getting the X and y data
X = high_ads.drop('CountryCode', axis=1)
y = high_ads['CountryCode']
# Getting the accuracy score
accuracy = (accuracy_score(y, pl.predict(X)))
scores['high'] = accuracy
scores
```

```
Out[28]: {'high': 0.7812646370023419, 'low': 0.7782466010314112}
```

Now we will perform a permutation test on the high and low data to see if there is a significant difference between the accuracies. I will define my test statistic as the difference in accuracy.

Null Hypothesis: The model predicts with the same accuracy for ads with high number of impressions as ads with low number of impressions.

Alternative Hypothesis: The model predicts with a different accuracy for ads with a high number of impressions as ads with low number of impressions.

Note: We define high number of impressions to be ads with impressions equal to or greater than the median impressions for all ads, and low number of impressions is ads with impressions less than the median.

Decision: We will set a significance level of .05. So we will reject the null hypothesis for a p value less than .05 and fail to reject the null hypothesis if it is greater than .05.

```
In [29]: obs = scores['high'] - scores['low']
         # adding a column called level which indicates if impressions is
         # low or high
         ads['level'] = ads['Impressions'].apply(
             lambda x: 'low' if x < med else 'high')


         n_repetitions = 100
         stats = []
         for i in range(n_repetitions):
             shuffled_impres = (
                 ads['Impressions']
                 .sample(replace=False, frac=1)
                 .reset_index(drop=True)
             )
```

```
            shuffled = (
                ads
                .assign(**{'Impressions': shuffled_impres})
            )
            # Dropping the level column now that we're done shuffling
            shuffled = shuffled.drop('level', axis=1)

            # Separating the data into low and high
            med = shuffled['Impressions'].median()
            low_ads = shuffled[shuffled['Impressions'] < med]
            high_ads = shuffled[shuffled['Impressions'] >= med]

            # Computing the accuracy for low
            X = low_ads.drop('CountryCode', axis=1)
            y = low_ads['CountryCode']

            # Getting the accuracy score
            accuracy = (accuracy_score(y, pl.predict(X)))
            scores['low'] = accuracy

            # Now find accuracy of high
            X = high_ads.drop('CountryCode', axis=1)
            y = high_ads['CountryCode']
            # Fitting the data
            pl.fit(X, y)
            # Getting the accuracy score
            accuracy = (accuracy_score(y, pl.predict(X)))
            scores['high'] = accuracy
            stats.append(scores['high'] - scores['low'])

        np.count_nonzero(stats >= obs) / n_repetitions
```

```
Out[29]: 0.61
```