Modeling-Linear-Reg

May 26, 2021

1 Modeling

The modeling phase of this project consists of 2 approaches of learning, in which we plan to test and grid search among the best and most accurate model. below we are going to develope the theoretical foundations of these algorithms:

- Regression, for regression we are using these machine learning algorithms, train the model based on the best hyperparameter combination and finally select the most accurate model.
  - multiple linear regression
  - polynomial regression
  - ridge
  - lasso
  - support vector machine
  - k-nearest neighbor
  - decision tree
  - random forest
- binary-class classification, we have 2 classification problems in this project, one binary classification, one multi class classification, implemented algorithms for classification will be these algorithms:
  - logistic regression
  - k-nearest neighbor
  - decision tree
  - support vector machine
  - gradient boosting
- for unsupervised learning and clusteirng approach of this project we will implement density-based clusteirng and DBSCAN algorithm.

in this notebook we are going to investigate the linear model regression and their most accurate hyperparameter and algorithm combination for our 4 datasets.

---

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import warnings
import matplotlib
warnings.filterwarnings("ignore")
```

```python
pd.set_option('display.max_rows', 200)
import seaborn as sns
from openpyxl import load_workbook
np.set_printoptions(suppress=True)
pd.set_option('display.float_format', lambda x: '%.2f' % x)
from sklearn import preprocessing
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from tqdm import tqdm
```

[2]:
```python
xls = pd.ExcelFile('data/Main Dataset V3.0 .xlsx')
ad_post = pd.read_excel(xls, 'Ad-Post')
ad_story = pd.read_excel(xls, 'Ad-Story')
influencer = pd.read_excel(xls, 'Influencer')
leaders_post = pd.read_excel(xls, 'Leaders-Post')
leaders_story = pd.read_excel(xls, 'Leaders-Story')
post = pd.read_excel(xls, 'Post')
story = pd.read_excel(xls, 'Story')
print('Datasets Loaded Completely.')
```

```
Datasets Loaded Completely.
```

since there are some categorical variables, we need to address them before using them for training the models based on them, since we are using multiple approaches thus the encoding approach can be different. for instance when we are using algorithms based on tree, label encoding is better than one hot encoding. on that circumstances, we are implementing different encoding and use the apropriate one for modeling technique.

[3]:
```python
#dummying dataset

# advertising posts
dummy_field = pd.get_dummies(ad_post['field'], prefix='field')
ad_post_dummy = pd.concat([ad_post, dummy_field], axis=1)
ad_post_dummy.drop(['field'], axis=1, inplace=True)

# advertising stories
dummy_field = pd.get_dummies(ad_story['field'], prefix='field')
ad_story_dummy = pd.concat([ad_story, dummy_field], axis=1)
ad_story_dummy.drop(['field'], axis=1, inplace=True)

#influencer
dummy_gender = pd.get_dummies(influencer['gender'], prefix='gender')
dummy_field = pd.get_dummies(influencer['field'], prefix='field')
influencer_dummy = pd.concat([influencer, dummy_gender, dummy_field], axis=1)
influencer_dummy.drop(['gender', 'field'], axis=1, inplace=True)

#leaders posts
dummy_gender = pd.get_dummies(leaders_post['gender'], prefix='gender')
leaders_post_dummy = pd.concat([leaders_post, dummy_gender], axis=1)
```

```
leaders_post_dummy.drop(['gender'], axis=1, inplace=True)
```

[4]:
```
# label encoding dataset

# advertising posts
labels, _ = pd.factorize(ad_post['field'])
ad_post_labelencoded = ad_post
ad_post_labelencoded['field_labelencoded'] = labels.tolist()

# advertising stories
labels, _ = pd.factorize(ad_story['field'])
ad_story_labelencoded = ad_story
ad_story_labelencoded['field_labelencoded'] = labels.tolist()

# influencer
labels, _ = pd.factorize(influencer['gender'])
influencer_labelencoded = influencer
influencer_labelencoded['gender_labelencoded'] = labels.tolist()
labels, _ = pd.factorize(influencer['field'])
influencer_labelencoded['field_labelencoded'] = labels.tolist()

# leaders post
labels, _ = pd.factorize(leaders_post['gender'])
leaders_post_labelencoded = leaders_post
leaders_post_labelencoded['gender_labelencoded'] = labels.tolist()
```

1.1 Regression

1.1.1 Multiple Linear Regression

Advertising Posts

[5]:
```
from sklearn.linear_model import LinearRegression
```

[6]:
```
ad_post_y = np.asarray(ad_post_dummy['cost'])
ad_post_x = np.asarray(ad_post_dummy.loc[:, ['follower', 'view', 'threshold',␣
↪'field_art & culture', 'field_fact', 'field_video', 'field_video']])
```

[7]:
```
temp_lst = []
for i in tqdm(range(2, 10)):
    kf = KFold(n_splits = i)
    for train_index, test_index in kf.split(ad_post_x):
        X_train, X_test = ad_post_x[train_index], ad_post_x[test_index]
        y_train, y_test = ad_post_y[train_index], ad_post_y[test_index]
        reg_lr_unnormalize = LinearRegression()
        reg_lr_normalize = LinearRegression(normalize=True)
        reg_lr_unnormalize.fit(X_train, y_train)
        reg_lr_normalize.fit(X_train, y_train)
        temp_lst2 = []
```

```
        temp_lst2.append(i)
        temp_lst2.append(reg_lr_unnormalize.score(X_train, y_train))
        temp_lst2.append(reg_lr_normalize.score(X_train, y_train))
        temp_lst2.append(reg_lr_unnormalize.score(X_test, y_test))
        temp_lst2.append(reg_lr_normalize.score(X_test, y_test))
        temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst, columns=['k', 'Unnormalized Train Score',␣
 ↪'Normalized Train Score', 'Unnormalized Test Score', 'Normalized Test␣
 ↪Score'])

temp_lst = []
for k in range(2, 10):
    temp_lst2 = []
    temp_lst2.append(k)
    temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] ==␣
 ↪k)]['Unnormalized Train Score']), decimals=4))
    temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k)]['Normalized␣
 ↪Train Score']), decimals=4))
    temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] ==␣
 ↪k)]['Unnormalized Test Score']), decimals=4))
    temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k)]['Normalized␣
 ↪Test Score']), decimals=4))
    temp_lst.append(temp_lst2)

reg_lr_eval_df = pd.DataFrame(temp_lst, columns=['k', 'Unnormalized Train␣
 ↪Score', 'Normalized Train Score', 'Unnormalized Test Score', 'Normalized␣
 ↪Test Score'])
reg_lr_eval_df
```

```
100%|      | 8/8 [00:00<00:00, 57.16it/s]
```

[7]:

```
   k  Unnormalized Train Score  Normalized Train Score  \
0  2                      0.95                    0.95
1  3                      0.93                    0.93
2  4                      0.93                    0.93
3  5                      0.93                    0.93
4  6                      0.92                    0.92
5  7                      0.92                    0.92
6  8                      0.92                    0.92
7  9                      0.92                    0.92

   Unnormalized Test Score  Normalized Test Score
0                     0.71                   0.71
1                     0.77                   0.77
2                     0.48                   0.48
3                     0.54                   0.54
```

```
4                      0.35                    0.35
5                      0.05                    0.05
6                     -0.04                   -0.04
7                     -3.50                   -3.50
```

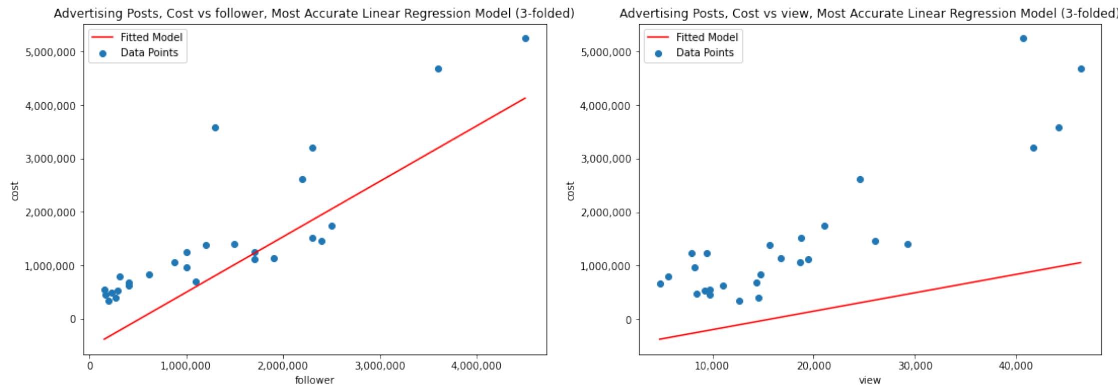
as you can see the best model for linear regression was 3-folded with accuracy of 77% in test dataset. in the cell below we are going to implement this as final model, also normalizing data wouldn't affect the overall accuracy so we dont normalize data.

[8]:
```
kf = KFold(n_splits = 3)
reg_lr = LinearRegression()
for train_index, test_index in kf.split(ad_post_x):
    X_train, X_test = ad_post_x[train_index], ad_post_x[test_index]
    y_train, y_test = ad_post_y[train_index], ad_post_y[test_index]
    reg_lr.fit(X_train, y_train)
```

[9]:
```
fig = plt.figure(figsize = (18, 6))
ax1 = fig.add_subplot(1,2,1)
ax2 = fig.add_subplot(1,2,2)

axs = [ax1, ax2]
feature_lst = ['follower', 'view']

for ax, feature in zip(axs, feature_lst):
    ax.scatter(ad_post_dummy[feature], ad_post_dummy['cost'], label='Data␣
↪Points')
    X_plot = np.arange(ad_post_dummy[feature].min(), ad_post_dummy[feature].
↪max() , 1)
    y_plot = reg_lr.coef_[axs.index(ax)] * X_plot + reg_lr.intercept_
    ax.plot(X_plot, y_plot, '-r', label='Fitted Model')
    ax.get_yaxis().set_major_formatter(matplotlib.ticker.FuncFormatter(lambda␣
↪x, p: format(int(x), ',')))
    ax.get_xaxis().set_major_formatter(matplotlib.ticker.FuncFormatter(lambda␣
↪x, p: format(int(x), ',')))
    ax.set_title(f'Advertising Posts, Cost vs {feature}, Most Accurate Linear␣
↪Regression Model (3-folded)')
    ax.set_xlabel(feature)
    ax.set_ylabel('cost')
    ax.legend()
plt.show()
```



Advertising Stories

[10]:
```
ad_story_y = np.asarray(ad_story_dummy['cost'])
ad_story_x = np.asarray(ad_story_dummy.loc[:, ['view', 'follower', 'action',␣
↪'interaction', 'impression', 'field_art & culture', 'field_fact',␣
↪'field_health',
                                              'field_news', 'field_video',␣
↪'field_women']])
```

[11]:
```
temp_lst = []
for i in tqdm(range(2, 10)):
    kf = KFold(n_splits = i)
    for train_index, test_index in kf.split(ad_story_x):
        X_train, X_test = ad_story_x[train_index], ad_story_x[test_index]
        y_train, y_test = ad_story_y[train_index], ad_story_y[test_index]
        reg_lr_unnormalize = LinearRegression()
        reg_lr_normalize = LinearRegression(normalize=True)
        reg_lr_unnormalize.fit(X_train, y_train)
        reg_lr_normalize.fit(X_train, y_train)
        temp_lst2 = []
        temp_lst2.append(i)
        temp_lst2.append(reg_lr_unnormalize.score(X_train, y_train))
        temp_lst2.append(reg_lr_normalize.score(X_train, y_train))
        temp_lst2.append(reg_lr_unnormalize.score(X_test, y_test))
        temp_lst2.append(reg_lr_normalize.score(X_test, y_test))
        temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst, columns=['k', 'Unnormalized Train Score',␣
↪'Normalized Train Score', 'Unnormalized Test Score', 'Normalized Test␣
↪Score'])

temp_lst = []
for k in range(2, 10):
```

```
    temp_lst2 = []
    temp_lst2.append(k)
    temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] ==␣
↪k)]['Unnormalized Train Score']), decimals=4))
    temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k)]['Normalized␣
↪Train Score']), decimals=4))
    temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] ==␣
↪k)]['Unnormalized Test Score']), decimals=4))
    temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k)]['Normalized␣
↪Test Score']), decimals=4))
    temp_lst.append(temp_lst2)

reg_lr_eval_df = pd.DataFrame(temp_lst, columns=['k', 'Unnormalized Train␣
↪Score', 'Normalized Train Score', 'Unnormalized Test Score', 'Normalized␣
↪Test Score'])
reg_lr_eval_df
```

```
100%|     | 8/8 [00:00<00:00, 75.64it/s]
```

[11]:

```
   k  Unnormalized Train Score  Normalized Train Score  \
0  2                      1.00                    1.00
1  3                      1.00                    1.00
2  4                      1.00                    1.00
3  5                      1.00                    1.00
4  6                      1.00                    1.00
5  7                      1.00                    1.00
6  8                      1.00                    1.00
7  9                      1.00                    1.00

   Unnormalized Test Score  Normalized Test Score
0                     0.92                   0.93
1                     0.96                   0.96
2                     0.91                   0.91
3                     0.91                   0.91
4                     0.62                   0.62
5                     0.80                   0.80
6                     0.79                   0.79
7                     0.75                   0.75
```

as you can see in the table above, the best performing and most accurate multiple linear regression is a 3-folded one with train accuracy of 100% and test accuracy of 96%. Thus we implement a model based on this circumstances and use it to predict further costs of advertising stories. like advertising posts, normalizing values in this model is not a deciding factor regarding the accuracy, so we don't normalize values.

[12]:

```
kf = KFold(n_splits = 3)
reg_lr = LinearRegression()
for train_index, test_index in kf.split(ad_post_x):
```

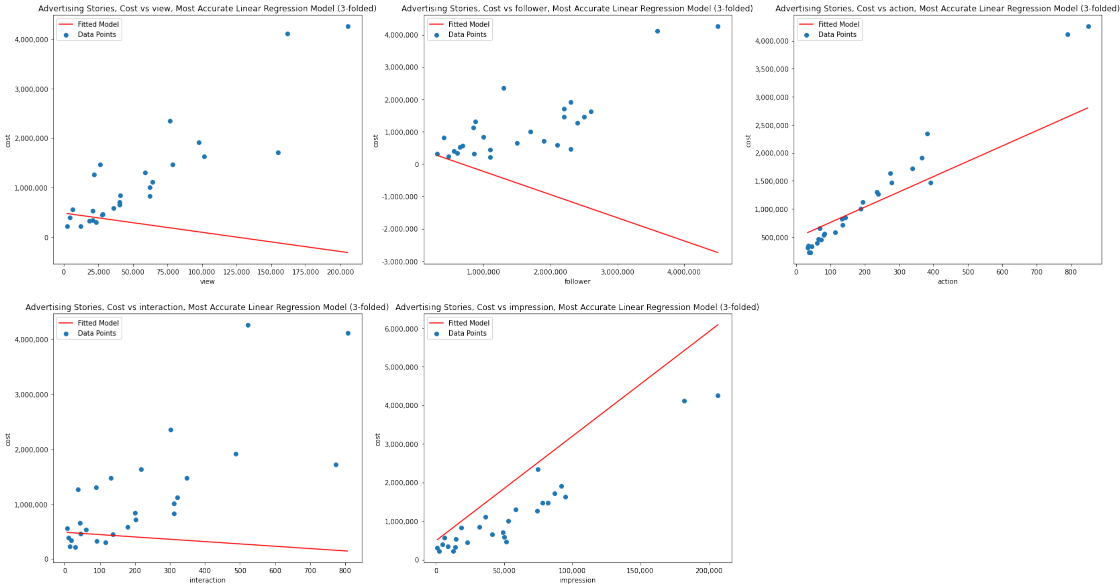
```
    X_train, X_test = ad_story_x[train_index], ad_story_x[test_index]
    y_train, y_test = ad_story_y[train_index], ad_story_y[test_index]
    reg_lr.fit(X_train, y_train)
```

[13]:
```
fig = plt.figure(figsize = (28, 15))
ax1 = fig.add_subplot(2,3,1)
ax2 = fig.add_subplot(2,3,2)
ax3 = fig.add_subplot(2,3,3)
ax4 = fig.add_subplot(2,3,4)
ax5 = fig.add_subplot(2,3,5)



axs = [ax1, ax2, ax3, ax4, ax5]
feature_lst = ['view', 'follower', 'action', 'interaction', 'impression']

for ax, feature in zip(axs, feature_lst):
    ax.scatter(ad_story_dummy[feature], ad_story_dummy['cost'], label='Data␣
↪Points')
    X_plot = np.arange(ad_story_dummy[feature].min(), ad_story_dummy[feature].
↪max() , 1)
    y_plot = reg_lr.coef_[axs.index(ax)] * X_plot + reg_lr.intercept_
    ax.plot(X_plot, y_plot, '-r', label='Fitted Model')
    ax.get_yaxis().set_major_formatter(matplotlib.ticker.FuncFormatter(lambda␣
↪x, p: format(int(x), ',')))
    ax.get_xaxis().set_major_formatter(matplotlib.ticker.FuncFormatter(lambda␣
↪x, p: format(int(x), ',')))
    ax.set_title(f'Advertising Stories, Cost vs {feature}, Most Accurate Linear␣
↪Regression Model (3-folded)')
    ax.set_xlabel(feature)
    ax.set_ylabel('cost')
    ax.legend()
plt.show()
```



Influencers

[14]:
```
influencer_y = np.asarray(influencer_dummy['cost'])
influencer_x = np.asarray(influencer_dummy.loc[:, ['follower', 'view',␣
 ↪'action', 'impression', 'cta', 'interaction', 'gender_family',␣
 ↪'gender_female',
                                                      'gender_male',␣
 ↪'field_cooking', 'field_health', 'field_lifestyle', 'field_sport',␣
 ↪'field_tourism']])
```

[15]:
```
temp_lst = []
for i in tqdm(range(2, 10)):
    kf = KFold(n_splits = i)
    for train_index, test_index in kf.split(influencer_x):
        X_train, X_test = influencer_x[train_index], influencer_x[test_index]
        y_train, y_test = influencer_y[train_index], influencer_y[test_index]
        reg_lr_unnormalize = LinearRegression()
        reg_lr_normalize = LinearRegression(normalize=True)
        reg_lr_unnormalize.fit(X_train, y_train)
        reg_lr_normalize.fit(X_train, y_train)
        temp_lst2 = []
        temp_lst2.append(i)
        temp_lst2.append(reg_lr_unnormalize.score(X_train, y_train))
        temp_lst2.append(reg_lr_normalize.score(X_train, y_train))
        temp_lst2.append(reg_lr_unnormalize.score(X_test, y_test))
        temp_lst2.append(reg_lr_normalize.score(X_test, y_test))
        temp_lst.append(temp_lst2)
```

```
temp_df = pd.DataFrame(temp_lst, columns=['k', 'Unnormalized Train Score',␣
↪'Normalized Train Score', 'Unnormalized Test Score', 'Normalized Test␣
↪Score'])

temp_lst = []
for k in range(2, 10):
    temp_lst2 = []
    temp_lst2.append(k)
    temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] ==␣
↪k)]['Unnormalized Train Score']), decimals=4))
    temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k)]['Normalized␣
↪Train Score']), decimals=4))
    temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] ==␣
↪k)]['Unnormalized Test Score']), decimals=4))
    temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k)]['Normalized␣
↪Test Score']), decimals=4))
    temp_lst.append(temp_lst2)

reg_lr_eval_df = pd.DataFrame(temp_lst, columns=['k', 'Unnormalized Train␣
↪Score', 'Normalized Train Score', 'Unnormalized Test Score', 'Normalized␣
↪Test Score'])
reg_lr_eval_df
```

```
100%|          | 8/8 [00:00<00:00, 55.70it/s]
```

[15]:

```
   k  Unnormalized Train Score  Normalized Train Score  \
0  2                      0.87                    0.87
1  3                      0.88                    0.88
2  4                      0.85                    0.85
3  5                      0.85                    0.85
4  6                      0.85                    0.85
5  7                      0.84                    0.84
6  8                      0.85                    0.85
7  9                      0.84                    0.84

   Unnormalized Test Score  Normalized Test Score
0                     0.22                   0.25
1                    -0.37                  -0.38
2                    -0.17                  -0.17
3                    -0.30                  -0.30
4                    -0.13                  -0.13
5                    -1.00                  -1.01
6                    -0.58                  -0.58
7                    -1.93                  -1.93
```

as you can see above, the multiple linear regression model is underfitted to the data to a very large extreme. this dataset is not acting linearly so linear regression wouldn't fit to it, thus this approach is not good for this situation. on that circumstances, we wont go any further with multiple linear regression for this dataset since it's not a appropriate fit.

Leaders Posts

[16]:
```
leaders_post_y = np.asarray(leaders_post_dummy['cost'])
leaders_post_x = np.asarray(leaders_post_dummy.loc[:, ['follower', 'view',␣
↪'like', 'comment', 'share', 'save', 'profile_visit', 'reach', 'impression',
                                                     'gender_family',␣
↪'gender_female', 'gender_male']])
```

[17]:
```
temp_lst = []
for i in tqdm(range(2, 10)):
    kf = KFold(n_splits = i)
    for train_index, test_index in kf.split(leaders_post_x):
        X_train, X_test = leaders_post_x[train_index],␣
↪leaders_post_x[test_index]
        y_train, y_test = leaders_post_y[train_index],␣
↪leaders_post_y[test_index]
        reg_lr_unnormalize = LinearRegression()
        reg_lr_normalize = LinearRegression(normalize=True)
        reg_lr_unnormalize.fit(X_train, y_train)
        reg_lr_normalize.fit(X_train, y_train)
        temp_lst2 = []
        temp_lst2.append(i)
        temp_lst2.append(reg_lr_unnormalize.score(X_train, y_train))
        temp_lst2.append(reg_lr_normalize.score(X_train, y_train))
        temp_lst2.append(reg_lr_unnormalize.score(X_test, y_test))
        temp_lst2.append(reg_lr_normalize.score(X_test, y_test))
        temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst, columns=['k', 'Unnormalized Train Score',␣
↪'Normalized Train Score', 'Unnormalized Test Score', 'Normalized Test␣
↪Score'])

temp_lst = []
for k in range(2, 10):
    temp_lst2 = []
    temp_lst2.append(k)
    temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] ==␣
↪k)]['Unnormalized Train Score']), decimals=4))
    temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k)]['Normalized␣
↪Train Score']), decimals=4))
    temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] ==␣
↪k)]['Unnormalized Test Score']), decimals=4))
    temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k)]['Normalized␣
↪Test Score']), decimals=4))
    temp_lst.append(temp_lst2)
```

```
reg_lr_eval_df = pd.DataFrame(temp_lst, columns=['k', 'Unnormalized Train␣
↪Score', 'Normalized Train Score', 'Unnormalized Test Score', 'Normalized␣
↪Test Score'])
reg_lr_eval_df
```

```
100%|      | 8/8 [00:00<00:00, 58.55it/s]
```

[17]:
```
   k  Unnormalized Train Score  Normalized Train Score  \
0  2                      1.00                    1.00
1  3                      1.00                    1.00
2  4                      1.00                    1.00
3  5                      1.00                    1.00
4  6                      1.00                    1.00
5  7                      1.00                    1.00
6  8                      1.00                    1.00
7  9                      1.00                    1.00

   Unnormalized Test Score  Normalized Test Score
0                  -119.27                  -3.67
1                   -51.77                -331.70
2               -200198.36                -673.05
3                 -1440.19                -988.99
4                 -1770.67               -1317.02
5                 -2655.51               -1974.55
6                   -52.77                  -1.06
7                      nan                    nan
```

as you can see, this dataset is underfitted to the multiple linear regression model too. the amount of underfitting is significance as you can see the difference between the training error and test error is very large. this indicates that multiple linear regression is not good fit for this dataset, the other reason for this behavior is also the small number of record available in this dataset.

1.1.2 Polynomial Regression

another approach of regression porblems is polynomial regression. we are going to use only one independent variable for predicting the dependent variable. the independent variable of choice is view since it's the main performance metric.

Advertising Posts

[18]:
```
from sklearn.preprocessing import PolynomialFeatures
```

[19]:
```
ad_post_x = np.asarray(ad_post[['view']])
ad_post_y = np.asarray(ad_post[['cost']])
```

[20]:
```
temp_lst = []
for i in tqdm(range(2, 10)):
    kf = KFold(n_splits = i)
```

```
    for train_index, test_index in kf.split(ad_post_x):
        X_train, X_test = ad_post_x[train_index], ad_post_x[test_index]
        y_train, y_test = ad_post_y[train_index], ad_post_y[test_index]
        for j in range(2, 6):
            poly = PolynomialFeatures(degree=j)
            X_train_poly = poly.fit_transform(X_train)
            X_test_poly = poly.fit_transform(X_test)
            reg_lr_unnormalize = LinearRegression()
            reg_lr_normalize = LinearRegression(normalize=True)
            reg_lr_unnormalize.fit(X_train_poly, y_train)
            reg_lr_normalize.fit(X_train_poly, y_train)
            temp_lst2 = []
            temp_lst2.append(j)
            temp_lst2.append(i)
            temp_lst2.append(reg_lr_unnormalize.score(X_train_poly, y_train))
            temp_lst2.append(reg_lr_normalize.score(X_train_poly, y_train))
            temp_lst2.append(reg_lr_unnormalize.score(X_test_poly, y_test))
            temp_lst2.append(reg_lr_normalize.score(X_test_poly, y_test))
            temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst, columns=['polynomial degree', 'k',␣
 ↪'Unnormalized Train Score', 'Normalized Train Score', 'Unnormalized Test␣
 ↪Score', 'Normalized Test Score'])

temp_lst = []
for k in range(2, 10):
    for c in range(2, 6):
        temp_lst2 = []
        temp_lst2.append(c)
        temp_lst2.append(k)
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['polynomial degree'] == c)]['Unnormalized Train Score']),␣
 ↪decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['polynomial degree'] == c)]['Normalized Train Score']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['polynomial degree'] == c)]['Unnormalized Test Score']),␣
 ↪decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['polynomial degree'] == c)]['Normalized Test Score']), decimals=4))
        temp_lst.append(temp_lst2)

reg_lr_eval_df = pd.DataFrame(temp_lst, columns=['polynomial degree', 'k',␣
 ↪'Unnormalized Train Score', 'Normalized Train Score', 'Unnormalized Test␣
 ↪Score', 'Normalized Test Score'])
reg_lr_eval_df
```

```
100%|          | 8/8 [00:00<00:00, 19.37it/s]
```

[20]:

	polynomial degree	k	Unnormalized Train Score	Normalized Train Score	\
0	2	2	0.91	0.91	
1	3	2	0.92	0.92	
2	4	2	0.94	0.95	
3	5	2	0.94	0.95	
4	2	3	0.86	0.86	
5	3	3	0.87	0.87	
6	4	3	0.89	0.90	
7	5	3	0.90	0.92	
8	2	4	0.85	0.85	
9	3	4	0.86	0.86	
10	4	4	0.87	0.87	
11	5	4	0.87	0.89	
12	2	5	0.85	0.85	
13	3	5	0.86	0.86	
14	4	5	0.87	0.87	
15	5	5	0.86	0.87	
16	2	6	0.85	0.85	
17	3	6	0.86	0.86	
18	4	6	0.86	0.86	
19	5	6	0.86	0.87	
20	2	7	0.84	0.84	
21	3	7	0.85	0.85	
22	4	7	0.86	0.86	
23	5	7	0.86	0.86	
24	2	8	0.84	0.84	
25	3	8	0.85	0.85	
26	4	8	0.86	0.86	
27	5	8	0.85	0.86	
28	2	9	0.84	0.84	
29	3	9	0.85	0.85	
30	4	9	0.85	0.85	
31	5	9	0.85	0.86	

	Unnormalized Test Score	Normalized Test Score
0	0.39	0.39
1	-0.05	-0.05
2	-0.44	-3.16
3	-0.92	0.18
4	0.70	0.70
5	0.62	0.62
6	0.37	0.29
7	0.32	-0.27
8	0.34	0.34
9	0.29	0.29

```
10                   0.17                 0.17
11                   0.15                -0.09
12                   0.51                 0.51
13                   0.36                 0.36
14                   0.26                 0.27
15                   0.30                -0.05
16                   0.31                 0.31
17                   0.30                 0.30
18                   0.20                 0.16
19                   0.17                -0.20
20                   0.28                 0.28
21                   0.13                 0.13
22                   0.07                 0.06
23                   0.11                -0.33
24                  -1.57                -1.57
25                  -1.24                -1.24
26                  -1.54                -1.53
27                  -1.74                -1.48
28                  -9.84                -9.84
29                  -9.60                -9.60
30                 -10.61               -10.27
31                  -9.85               -13.79
```

as you can see in the table above, the most accurate polynomial linear regression model is 3-folded second degree polynomial model which scored 86% in training dataset and 70& in test dataset. it's worthy to mention that simple 3-folded multiple linear regression managed to achieve 93% in training set and 77% in test dataset. since this performance is not very bad, it's worth to plot the learned equation on dataset.

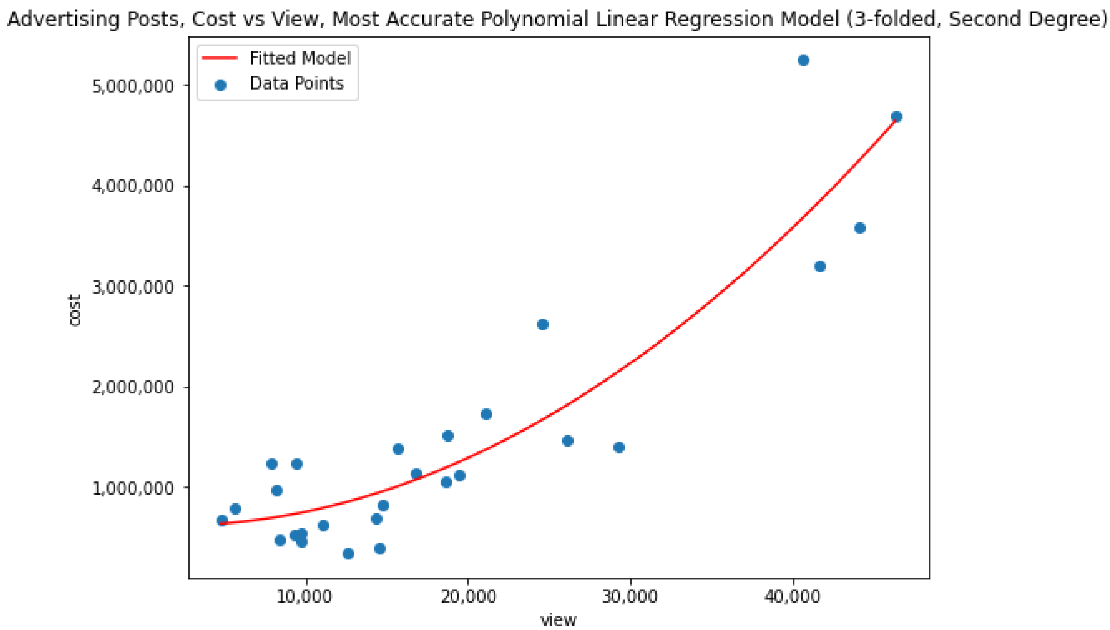
[21]:
```python
kf = KFold(n_splits = 3)
reg_lr = LinearRegression()
poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(ad_post_x)
reg_lr.fit(X_train_poly, ad_post_y)
```

[21]:
```
LinearRegression()
```

[22]:
```python
fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot()

ax.scatter(ad_post_dummy['view'], ad_post_dummy['cost'], label='Data Points')
X_plot = np.arange(ad_post_dummy['view'].min(), ad_post_dummy['view'].max() , 1)
y_plot = reg_lr.intercept_[0] + reg_lr.coef_[0][1]*X_plot+ reg_lr.
 ↪coef_[0][2]*np.power(X_plot, 2)
ax.plot(X_plot, y_plot, '-r', label='Fitted Model')
ax.get_yaxis().set_major_formatter(matplotlib.ticker.FuncFormatter(lambda x, p:␣
 ↪format(int(x), ',')))
```

```
ax.get_xaxis().set_major_formatter(matplotlib.ticker.FuncFormatter(lambda x, p:␣
↪format(int(x), ',')))
ax.set_title(f'Advertising Posts, Cost vs View, Most Accurate Polynomial Linear␣
↪Regression Model (3-folded, Second Degree)')
ax.set_xlabel('view')
ax.set_ylabel('cost')
ax.legend()
plt.show()
```



Advertising stories

[23]:
```
ad_story_x = np.asarray(ad_story[['view']])
ad_story_y = np.asarray(ad_story[['cost']])
```

[24]:
```
temp_lst = []
for i in tqdm(range(2, 10)):
    kf = KFold(n_splits = i)
    for train_index, test_index in kf.split(ad_story_x):
        X_train, X_test = ad_story_x[train_index], ad_story_x[test_index]
        y_train, y_test = ad_story_y[train_index], ad_story_y[test_index]
        for j in range(2, 6):
            poly = PolynomialFeatures(degree=j)
            X_train_poly = poly.fit_transform(X_train)
            X_test_poly = poly.fit_transform(X_test)
            reg_lr_unnormalize = LinearRegression()
            reg_lr_normalize = LinearRegression(normalize=True)
```

```
                reg_lr_unnormalize.fit(X_train_poly, y_train)
                reg_lr_normalize.fit(X_train_poly, y_train)
                temp_lst2 = []
                temp_lst2.append(j)
                temp_lst2.append(i)
                temp_lst2.append(reg_lr_unnormalize.score(X_train_poly, y_train))
                temp_lst2.append(reg_lr_normalize.score(X_train_poly, y_train))
                temp_lst2.append(reg_lr_unnormalize.score(X_test_poly, y_test))
                temp_lst2.append(reg_lr_normalize.score(X_test_poly, y_test))
                temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst, columns=['polynomial degree', 'k',␣
 ↪'Unnormalized Train Score', 'Normalized Train Score', 'Unnormalized Test␣
 ↪Score', 'Normalized Test Score'])

temp_lst = []
for k in range(2, 10):
    for c in range(2, 6):
        temp_lst2 = []
        temp_lst2.append(c)
        temp_lst2.append(k)
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['polynomial degree'] == c)]['Unnormalized Train Score']),␣
 ↪decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['polynomial degree'] == c)]['Normalized Train Score']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['polynomial degree'] == c)]['Unnormalized Test Score']),␣
 ↪decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['polynomial degree'] == c)]['Normalized Test Score']), decimals=4))
        temp_lst.append(temp_lst2)

reg_lr_eval_df = pd.DataFrame(temp_lst, columns=['polynomial degree', 'k',␣
 ↪'Unnormalized Train Score', 'Normalized Train Score', 'Unnormalized Test␣
 ↪Score', 'Normalized Test Score'])
reg_lr_eval_df
```

100%| | 8/8 [00:00<00:00, 20.20it/s]

[24]:

	polynomial degree	k	Unnormalized Train Score	Normalized Train Score	\
0	2	2	0.81	0.81	
1	3	2	0.82	0.82	
2	4	2	0.86	0.87	
3	5	2	0.89	0.92	
4	2	3	0.81	0.81	
5	3	3	0.81	0.81	

6	4 3	0.83	0.84
7	5 3	0.86	0.88
8	2 4	0.81	0.81
9	3 4	0.82	0.82
10	4 4	0.85	0.86
11	5 4	0.87	0.88
12	2 5	0.81	0.81
13	3 5	0.81	0.81
14	4 5	0.82	0.83
15	5 5	0.83	0.85
16	2 6	0.81	0.81
17	3 6	0.81	0.81
18	4 6	0.82	0.82
19	5 6	0.82	0.85
20	2 7	0.81	0.81
21	3 7	0.81	0.81
22	4 7	0.82	0.82
23	5 7	0.82	0.84
24	2 8	0.81	0.81
25	3 8	0.82	0.82
26	4 8	0.83	0.83
27	5 8	0.83	0.85
28	2 9	0.81	0.81
29	3 9	0.82	0.82
30	4 9	0.83	0.83
31	5 9	0.83	0.84

	Unnormalized Test Score	Normalized Test Score
0	0.80	0.80
1	0.71	0.71
2	-15.67	-29.62
3	-89.33	-202.46
4	0.74	0.74
5	0.72	0.72
6	-6.75	-18.31
7	-62.54	-155.78
8	0.41	0.41
9	0.21	0.21
10	-5.87	-16.59
11	-56.68	-119.33
12	0.42	0.42
13	0.39	0.39
14	-5.56	-16.35
15	-56.60	-118.90
16	-0.45	-0.45
17	-0.68	-0.68
18	-1.27	-3.33

19	-18.09	-85.26
20	0.63	0.63
21	0.59	0.59
22	-0.07	-1.86
23	-14.46	-71.02
24	-0.23	-0.23
25	-0.61	-0.61
26	-0.86	-2.79
27	-13.10	-60.83
28	-0.27	-0.27
29	-0.59	-0.59
30	-2.29	-6.78
31	-31.04	-104.67

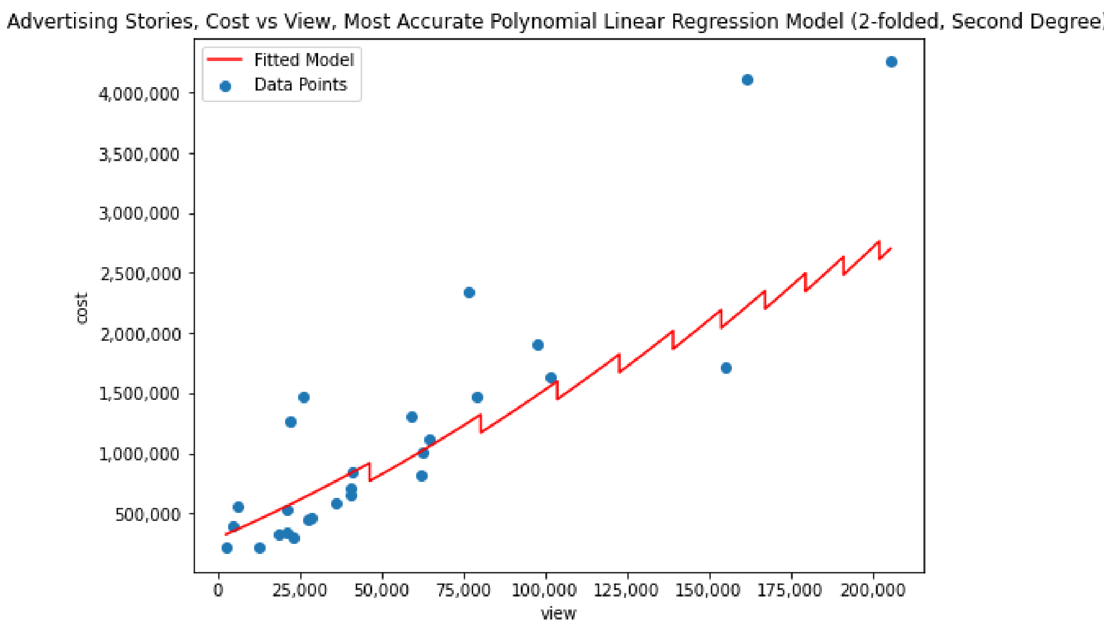
as you can see in the table above the best performing and most accurate polynomial linear regression model is 2-folded second degree with 80% accuracy. although this models is not better performing than multiple linear regression, since it has mediocre performance, it's good to plot it.

[25]:
```
kf = KFold(n_splits = 2)
reg_lr = LinearRegression()
poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(ad_story_x)
reg_lr.fit(X_train_poly, ad_story_y)
```

[25]:
```
LinearRegression()
```

[26]:
```
fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot()

ax.scatter(ad_story_dummy['view'], ad_story_dummy['cost'], label='Data Points')
X_plot = np.arange(ad_story_dummy['view'].min(), ad_story_dummy['view'].max() ,␣
 ↪1)
y_plot = reg_lr.intercept_[0] + reg_lr.coef_[0][1]*X_plot+ reg_lr.
 ↪coef_[0][2]*np.power(X_plot, 2)
ax.plot(X_plot, y_plot, '-r', label='Fitted Model')
ax.get_yaxis().set_major_formatter(matplotlib.ticker.FuncFormatter(lambda x, p:␣
 ↪format(int(x), ',')))
ax.get_xaxis().set_major_formatter(matplotlib.ticker.FuncFormatter(lambda x, p:␣
 ↪format(int(x), ',')))
ax.set_title(f'Advertising Stories, Cost vs View, Most Accurate Polynomial␣
 ↪Linear Regression Model (2-folded, Second Degree)')
ax.set_xlabel('view')
ax.set_ylabel('cost')
ax.legend()
plt.show()
```



Influencer & Leaders Posts

[27]:
```
influencer_x = np.asarray(influencer[['view']])
influencer_y = np.asarray(influencer[['cost']])
leaders_post_x = np.asarray(leaders_post[['view']])
leaders_post_y = np.asarray(leaders_post[['cost']])
```

[28]:
```
temp_lst = []
for i in tqdm(range(2, 10)):
    kf = KFold(n_splits = i)
    for train_index, test_index in kf.split(influencer_x):
        X_train, X_test = influencer_x[train_index], influencer_x[test_index]
        y_train, y_test = influencer_y[train_index], influencer_y[test_index]
        for j in range(2, 6):
            poly = PolynomialFeatures(degree=j)
            X_train_poly = poly.fit_transform(X_train)
            X_test_poly = poly.fit_transform(X_test)
            reg_lr_unnormalize = LinearRegression()
            reg_lr_normalize = LinearRegression(normalize=True)
            reg_lr_unnormalize.fit(X_train_poly, y_train)
            reg_lr_normalize.fit(X_train_poly, y_train)
            temp_lst2 = []
            temp_lst2.append(j)
            temp_lst2.append(i)
            temp_lst2.append(reg_lr_unnormalize.score(X_train_poly, y_train))
            temp_lst2.append(reg_lr_normalize.score(X_train_poly, y_train))
            temp_lst2.append(reg_lr_unnormalize.score(X_test_poly, y_test))
```

```
            temp_lst2.append(reg_lr_normalize.score(X_test_poly, y_test))
            temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst, columns=['polynomial degree', 'k',␣
 ↪'Unnormalized Train Score', 'Normalized Train Score', 'Unnormalized Test␣
 ↪Score', 'Normalized Test Score'])

temp_lst = []
for k in range(2, 10):
    for c in range(2, 6):
        temp_lst2 = []
        temp_lst2.append(c)
        temp_lst2.append(k)
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['polynomial degree'] == c)]['Unnormalized Train Score']),␣
 ↪decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['polynomial degree'] == c)]['Normalized Train Score']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['polynomial degree'] == c)]['Unnormalized Test Score']),␣
 ↪decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['polynomial degree'] == c)]['Normalized Test Score']), decimals=4))
        temp_lst.append(temp_lst2)

reg_lr_eval_df = pd.DataFrame(temp_lst, columns=['polynomial degree', 'k',␣
 ↪'Unnormalized Train Score', 'Normalized Train Score', 'Unnormalized Test␣
 ↪Score', 'Normalized Test Score'])
reg_lr_eval_df
```

```
100%|      | 8/8 [00:00<00:00, 16.37it/s]
```

[28]:

```
    polynomial degree  k  Unnormalized Train Score  Normalized Train Score  \
0                   2  2                      0.63                    0.63
1                   3  2                      0.69                    0.69
2                   4  2                      0.69                    0.70
3                   5  2                      0.71                    0.79
4                   2  3                      0.68                    0.68
5                   3  3                      0.75                    0.75
6                   4  3                      0.74                    0.76
7                   5  3                      0.71                    0.76
8                   2  4                      0.66                    0.66
9                   3  4                      0.70                    0.70
10                  4  4                      0.71                    0.72
11                  5  4                      0.66                    0.73
12                  2  5                      0.66                    0.66
13                  3  5                      0.69                    0.69
```

14	4	5	0.70	0.71
15	5	5	0.65	0.71
16	2	6	0.66	0.66
17	3	6	0.68	0.68
18	4	6	0.70	0.70
19	5	6	0.64	0.71
20	2	7	0.66	0.66
21	3	7	0.68	0.68
22	4	7	0.69	0.69
23	5	7	0.62	0.71
24	2	8	0.66	0.66
25	3	8	0.67	0.67
26	4	8	0.68	0.69
27	5	8	0.62	0.70
28	2	9	0.66	0.66
29	3	9	0.66	0.66
30	4	9	0.68	0.68
31	5	9	0.62	0.70

	Unnormalized Test Score	Normalized Test Score
0	-2.24	-2.24
1	-591.50	-591.50
2	-169.10	-162.66
3	-64832.53	-2873109.93
4	-1.45	-1.45
5	-432.58	-432.58
6	-8739.05	-8680.69
7	-4672.21	-62193.18
8	-0.83	-0.83
9	-370.84	-370.84
10	-7629.78	-7488.06
11	-5454.55	-34605.21
12	-0.40	-0.40
13	-398.37	-398.37
14	-9187.13	-9276.71
15	-11468.00	-6347.08
16	-0.03	-0.03
17	-471.19	-471.19
18	-11931.99	-11262.59
19	-15340.11	-15097.47
20	-0.96	-0.96
21	-490.94	-490.94
22	-10913.72	-10599.54
23	-11128.87	-9925.27
24	-0.57	-0.57
25	-0.82	-0.82
26	-0.57	-0.77

```
27                  -0.88               -0.53
28                  -3.11               -3.11
29                  -4.19               -4.19
30                  -3.22               -3.79
31                  -5.65               -3.46
```

[29]:

```
temp_lst = []
for i in tqdm(range(2, 10)):
    kf = KFold(n_splits = i)
    for train_index, test_index in kf.split(leaders_post_x):
        X_train, X_test = leaders_post_x[train_index],␣
 ↪leaders_post_x[test_index]
        y_train, y_test = leaders_post_y[train_index],␣
 ↪leaders_post_y[test_index]
        for j in range(2, 6):
            poly = PolynomialFeatures(degree=j)
            X_train_poly = poly.fit_transform(X_train)
            X_test_poly = poly.fit_transform(X_test)
            reg_lr_unnormalize = LinearRegression()
            reg_lr_normalize = LinearRegression(normalize=True)
            reg_lr_unnormalize.fit(X_train_poly, y_train)
            reg_lr_normalize.fit(X_train_poly, y_train)
            temp_lst2 = []
            temp_lst2.append(j)
            temp_lst2.append(i)
            temp_lst2.append(reg_lr_unnormalize.score(X_train_poly, y_train))
            temp_lst2.append(reg_lr_normalize.score(X_train_poly, y_train))
            temp_lst2.append(reg_lr_unnormalize.score(X_test_poly, y_test))
            temp_lst2.append(reg_lr_normalize.score(X_test_poly, y_test))
            temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst, columns=['polynomial degree', 'k',␣
 ↪'Unnormalized Train Score', 'Normalized Train Score', 'Unnormalized Test␣
 ↪Score', 'Normalized Test Score'])

temp_lst = []
for k in range(2, 10):
    for c in range(2, 6):
        temp_lst2 = []
        temp_lst2.append(c)
        temp_lst2.append(k)
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['polynomial degree'] == c)]['Unnormalized Train Score']),␣
 ↪decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['polynomial degree'] == c)]['Normalized Train Score']), decimals=4))
```

```
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['polynomial degree'] == c)]['Unnormalized Test Score']),␣
↪decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['polynomial degree'] == c)]['Normalized Test Score']), decimals=4))
        temp_lst.append(temp_lst2)

reg_lr_eval_df = pd.DataFrame(temp_lst, columns=['polynomial degree', 'k',␣
↪'Unnormalized Train Score', 'Normalized Train Score', 'Unnormalized Test␣
↪Score', 'Normalized Test Score'])
reg_lr_eval_df
```

```
100%|          | 8/8 [00:00<00:00, 19.56it/s]
```

[29]:

	polynomial degree	k	Unnormalized Train Score	Normalized Train Score	\
0	2	2	0.57	0.57	
1	3	2	0.88	0.88	
2	4	2	0.98	1.00	
3	5	2	1.00	1.00	
4	2	3	0.45	0.45	
5	3	3	0.75	0.75	
6	4	3	0.80	0.87	
7	5	3	0.81	1.00	
8	2	4	0.42	0.42	
9	3	4	0.60	0.60	
10	4	4	0.65	0.75	
11	5	4	0.69	0.84	
12	2	5	0.44	0.44	
13	3	5	0.46	0.46	
14	4	5	0.52	0.61	
15	5	5	0.54	0.74	
16	2	6	0.43	0.43	
17	3	6	0.44	0.44	
18	4	6	0.48	0.56	
19	5	6	0.52	0.69	
20	2	7	0.42	0.42	
21	3	7	0.43	0.43	
22	4	7	0.47	0.54	
23	5	7	0.50	0.65	
24	2	8	0.41	0.41	
25	3	8	0.43	0.43	
26	4	8	0.47	0.53	
27	5	8	0.52	0.67	
28	2	9	0.41	0.41	
29	3	9	0.43	0.43	
30	4	9	0.46	0.50	
31	5	9	0.51	0.63	

```
    Unnormalized Test Score  Normalized Test Score
0                     -0.42                  -0.42
1                    -81.40                 -81.40
2                  -3018.53                -196.74
3                -166901.68                -414.48
4                     -1.23                  -1.23
5                    -34.79                 -34.79
6                    -52.14                 -39.98
7                    -72.69             -116315.32
8                   -333.36                -333.36
9                   -500.46                -500.46
10                  -562.93                -582.99
11                  -580.21               -87406.45
12                    -5.14                  -5.14
13                    -4.18                  -4.18
14                    -4.88                 -46.37
15                    -5.68               -25644.41
16                    -6.87                  -6.87
17                    -5.22                  -5.22
18                    -5.41                 -60.73
19                    -6.50               -34189.67
20                    -9.77                  -9.77
21                    -7.27                  -7.27
22                    -7.35                 -90.18
23                    -9.06               -51282.70
24                     0.40                   0.40
25                     0.37                   0.37
26                     0.27                  -7.63
27                    -2.68                 -38.65
28                      nan                    nan
29                      nan                    nan
30                      nan                    nan
31                      nan                    nan
```

like multiple linear regression, polynomial regression is also not a good fit for influencers and leaders posts dataset, so we omit training a model for them based on this algorithms

---

1.1.3 Ridge & Lasso Regression

another regression we are going to investigate is ridge regression. these two algorithms have alpha as hyperparameter. this hyperparameter defines the strength of regularization for these algorithm, in order to find the optimal value for it we are going to use grid search technique.

[30]:
```python
from sklearn.linear_model import Ridge, Lasso
```

[31]:
```
ad_post_y = np.asarray(ad_post_dummy[['cost']])
ad_post_x = np.asarray(ad_post_dummy[['follower', 'view', 'field_art &␣
↪culture', 'field_fact', 'field_video', 'field_women']])

ad_story_y = np.asarray(ad_story_dummy[['cost']])
ad_story_x = np.asarray(ad_story_dummy[['view', 'follower', 'action',␣
↪'interaction', 'impression', 'field_art & culture', 'field_fact',␣
↪'field_health',
                                       'field_news', 'field_video',␣
↪'field_women']])

influencer_y = np.asarray(influencer_dummy[['cost']])
influencer_x = np.asarray(influencer_dummy[['follower', 'view', 'action',␣
↪'impression', 'cta', 'interaction', 'gender_family', 'gender_female',␣
↪'gender_male',
                                           'field_cooking', 'field_health',␣
↪'field_lifestyle', 'field_sport', 'field_tourism']])

leaders_post_y = np.asarray(leaders_post_dummy[['cost']])
leaders_post_x = np.asarray(leaders_post_dummy[['follower', 'view', 'like',␣
↪'comment', 'share', 'save', 'profile_visit', 'reach', 'impression',␣
↪'gender_family',
                                               'gender_female', 'gender_male']])
```

Advertising Posts

[32]:
```
temp_lst = []
alphas = np.linspace(0, 1, 10)
for i in tqdm(range(2, 10)):
    kf = KFold(n_splits = i)
    for train_index, test_index in kf.split(ad_post_x):
        X_train, X_test = ad_post_x[train_index], ad_post_x[test_index]
        y_train, y_test = ad_post_y[train_index], ad_post_y[test_index]
        for a in alphas:
            ridge_reg = Ridge(alpha=a)
            lasso_reg = Lasso(alpha=a)
            ridge_reg.fit(X_train, y_train)
            lasso_reg.fit(X_train, y_train)
            temp_lst2 = []
            temp_lst2.append(i)
            temp_lst2.append(a)
            temp_lst2.append(ridge_reg.score(X_train, y_train))
            temp_lst2.append(lasso_reg.score(X_train, y_train))
            temp_lst2.append(ridge_reg.score(X_test, y_test))
            temp_lst2.append(lasso_reg.score(X_test, y_test))
            temp_lst.append(temp_lst2)
```

```python
temp_df = pd.DataFrame(temp_lst, columns=['k', 'alpha', 'Ridge Train Score',␣
↪'Lasso Train Score', 'Ridge Test Score', 'Lasso Test Score'])

temp_lst = []
for k in range(2, 10):
    for al in alphas:
        temp_lst2 = []
        temp_lst2.append(k)
        temp_lst2.append(al)
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['alpha'] == al)]['Ridge Train Score']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['alpha'] == al)]['Lasso Train Score']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['alpha'] == al)]['Ridge Test Score']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['alpha'] == al)]['Lasso Test Score']), decimals=4))
        temp_lst.append(temp_lst2)

ridge_lasso_reg_eval_df = pd.DataFrame(temp_lst, columns=['k', 'alpha', 'Ridge␣
↪Train Score', 'Lasso Train Score', 'Ridge Test Score', 'Lasso Test Score'])
ridge_lasso_reg_eval_df
```

100%| | 8/8 [00:00<00:00, 8.18it/s]

[32]:

	k	alpha	Ridge Train Score	Lasso Train Score	Ridge Test Score	\
0	2	0.00	0.95	0.95	0.74	
1	2	0.11	0.95	0.95	0.74	
2	2	0.22	0.95	0.95	0.74	
3	2	0.33	0.95	0.95	0.74	
4	2	0.44	0.95	0.95	0.74	
5	2	0.56	0.95	0.95	0.74	
6	2	0.67	0.94	0.95	0.74	
7	2	0.78	0.94	0.95	0.74	
8	2	0.89	0.94	0.95	0.74	
9	2	1.00	0.94	0.95	0.74	
10	3	0.00	0.93	0.93	0.77	
11	3	0.11	0.93	0.93	0.77	
12	3	0.22	0.93	0.93	0.77	
13	3	0.33	0.93	0.93	0.77	
14	3	0.44	0.92	0.93	0.77	
15	3	0.56	0.92	0.93	0.77	
16	3	0.67	0.92	0.93	0.77	
17	3	0.78	0.92	0.93	0.77	
18	3	0.89	0.92	0.93	0.77	
19	3	1.00	0.92	0.93	0.77	
20	4	0.00	0.93	0.93	0.48	

21	4	0.11	0.93	0.93	0.50
22	4	0.22	0.93	0.93	0.51
23	4	0.33	0.93	0.93	0.52
24	4	0.44	0.93	0.93	0.53
25	4	0.56	0.92	0.93	0.53
26	4	0.67	0.92	0.93	0.54
27	4	0.78	0.92	0.93	0.54
28	4	0.89	0.92	0.93	0.54
29	4	1.00	0.92	0.93	0.54
30	5	0.00	0.93	0.93	0.54
31	5	0.11	0.93	0.93	0.55
32	5	0.22	0.92	0.93	0.56
33	5	0.33	0.92	0.93	0.56
34	5	0.44	0.92	0.93	0.56
35	5	0.56	0.92	0.93	0.56
36	5	0.67	0.92	0.93	0.56
37	5	0.78	0.92	0.93	0.56
38	5	0.89	0.92	0.93	0.56
39	5	1.00	0.92	0.93	0.56
40	6	0.00	0.92	0.92	0.35
41	6	0.11	0.92	0.92	0.38
42	6	0.22	0.92	0.92	0.41
43	6	0.33	0.92	0.92	0.42
44	6	0.44	0.92	0.92	0.43
45	6	0.56	0.92	0.92	0.44
46	6	0.67	0.92	0.92	0.44
47	6	0.78	0.92	0.92	0.45
48	6	0.89	0.92	0.92	0.45
49	6	1.00	0.92	0.92	0.45
50	7	0.00	0.92	0.92	0.05
51	7	0.11	0.92	0.92	0.15
52	7	0.22	0.92	0.92	0.22
53	7	0.33	0.92	0.92	0.27
54	7	0.44	0.92	0.92	0.31
55	7	0.56	0.92	0.92	0.33
56	7	0.67	0.92	0.92	0.36
57	7	0.78	0.92	0.92	0.37
58	7	0.89	0.92	0.92	0.38
59	7	1.00	0.92	0.92	0.39
60	8	0.00	0.92	0.92	-0.04
61	8	0.11	0.92	0.92	0.05
62	8	0.22	0.92	0.92	0.11
63	8	0.33	0.92	0.92	0.15
64	8	0.44	0.92	0.92	0.18
65	8	0.56	0.92	0.92	0.21
66	8	0.67	0.92	0.92	0.22
67	8	0.78	0.92	0.92	0.23

68	8	0.89	0.92	0.92	0.24
69	8	1.00	0.92	0.92	0.24
70	9	0.00	0.92	0.92	-3.50
71	9	0.11	0.92	0.92	-3.63
72	9	0.22	0.92	0.92	-3.77
73	9	0.33	0.92	0.92	-3.91
74	9	0.44	0.92	0.92	-4.06
75	9	0.56	0.92	0.92	-4.20
76	9	0.67	0.92	0.92	-4.34
77	9	0.78	0.92	0.92	-4.47
78	9	0.89	0.92	0.92	-4.59
79	9	1.00	0.92	0.92	-4.70

	Lasso Test Score
0	0.74
1	0.74
2	0.74
3	0.74
4	0.74
5	0.74
6	0.74
7	0.74
8	0.74
9	0.74
10	0.80
11	0.80
12	0.80
13	0.80
14	0.80
15	0.80
16	0.80
17	0.80
18	0.80
19	0.80
20	0.48
21	0.48
22	0.48
23	0.48
24	0.48
25	0.48
26	0.48
27	0.48
28	0.48
29	0.48
30	0.54
31	0.54
32	0.54

33	0.54
34	0.54
35	0.54
36	0.54
37	0.54
38	0.54
39	0.54
40	0.35
41	0.35
42	0.35
43	0.35
44	0.35
45	0.35
46	0.35
47	0.35
48	0.35
49	0.35
50	0.05
51	0.05
52	0.05
53	0.05
54	0.05
55	0.05
56	0.05
57	0.05
58	0.05
59	0.05
60	-0.04
61	-0.04
62	-0.04
63	-0.04
64	-0.04
65	-0.04
66	-0.04
67	-0.04
68	-0.04
69	-0.04
70	-3.50
71	-3.50
72	-3.50
73	-3.50
74	-3.50
75	-3.50
76	-3.50
77	-3.50
78	-3.50
79	-3.50

[33]:
```
ridge_lasso_reg_eval_df.nlargest(3, 'Ridge Test Score')
```

[33]:
```
    k  alpha  Ridge Train Score  Lasso Train Score  Ridge Test Score  \
11  3   0.11               0.93               0.93              0.77
12  3   0.22               0.93               0.93              0.77
13  3   0.33               0.93               0.93              0.77

    Lasso Test Score
11              0.80
12              0.80
13              0.80
```

[34]:
```
ridge_lasso_reg_eval_df.nlargest(3, 'Lasso Test Score')
```

[34]:
```
    k  alpha  Ridge Train Score  Lasso Train Score  Ridge Test Score  \
10  3   0.00               0.93               0.93              0.77
11  3   0.11               0.93               0.93              0.77
12  3   0.22               0.93               0.93              0.77

    Lasso Test Score
10              0.80
11              0.80
12              0.80
```

as you can see the top performing lasso and ridge regression algorithms are 3 folded and alpha value in them don't matter. also lasso is a better performing algorithm than ridge in this dataset. in the next cell we are going to train the most accurate model based on these hyperparameters and plot the trained equation on dataset.

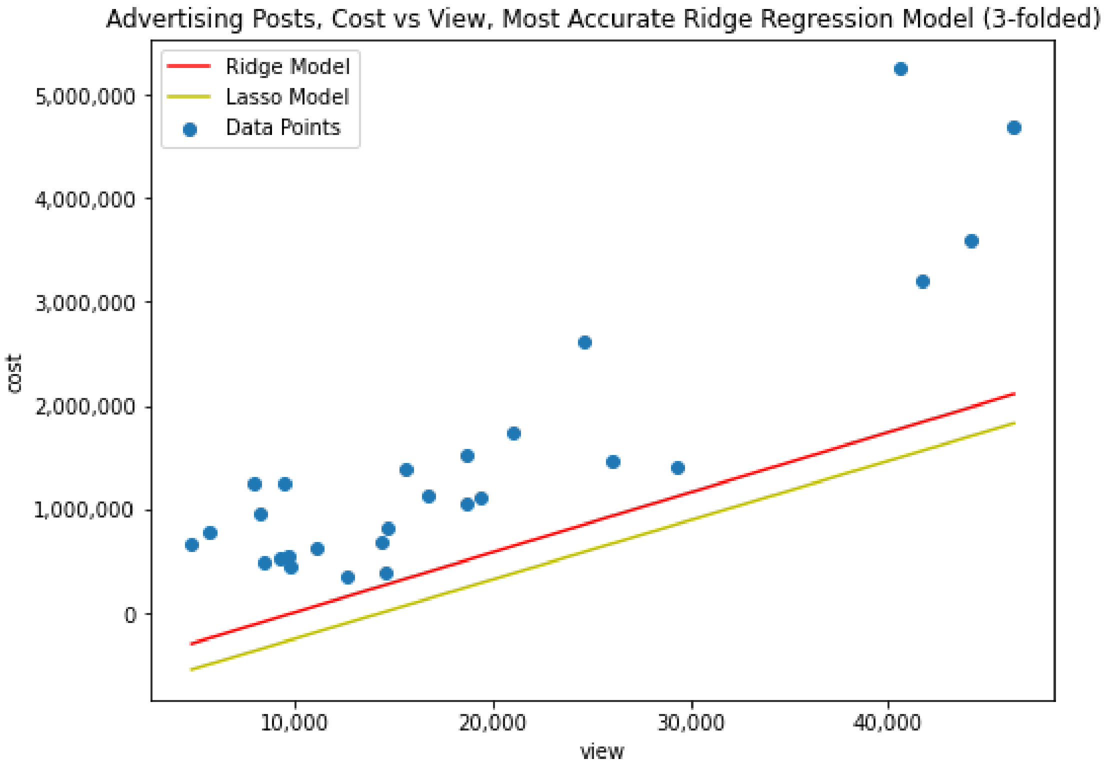
[35]:
```
kf = KFold(n_splits = 3)
ridge = Ridge()
lasso = Lasso()
ridge.fit(ad_post_x, ad_post_y)
lasso.fit(ad_post_x, ad_post_y)
```

[35]:
```
Lasso()
```

[36]:
```
fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot()

ax.scatter(ad_post_dummy['view'], ad_post_dummy['cost'], label='Data Points')
X_plot = np.arange(ad_post_dummy['view'].min(), ad_post_dummy['view'].max() , 1)
y_plot_ridge = ridge.intercept_[0] + (ridge.coef_[0][1]*X_plot)
y_plot_lasso = lasso.intercept_[0] + lasso.coef_[1] * X_plot
ax.plot(X_plot, y_plot_ridge, '-r', label='Ridge Model')
ax.plot(X_plot, y_plot_lasso, '-y', label='Lasso Model')
ax.get_yaxis().set_major_formatter(matplotlib.ticker.FuncFormatter(lambda x, p:␣
↪format(int(x), ',')))
```

```
ax.get_xaxis().set_major_formatter(matplotlib.ticker.FuncFormatter(lambda x, p:␣
↪format(int(x), ',')))
ax.set_title(f'Advertising Posts, Cost vs View, Most Accurate Ridge Regression␣
↪Model (3-folded)')
ax.set_xlabel('view')
ax.set_ylabel('cost')
ax.legend()
plt.show()
```



Advertising Stories

[37]:
```
temp_lst = []
alphas = np.linspace(0, 1, 10)
for i in tqdm(range(2, 10)):
    kf = KFold(n_splits = i)
    for train_index, test_index in kf.split(ad_story_x):
        X_train, X_test = ad_story_x[train_index], ad_story_x[test_index]
        y_train, y_test = ad_story_y[train_index], ad_story_y[test_index]
        for a in alphas:
            ridge_reg = Ridge(alpha=a)
            lasso_reg = Lasso(alpha=a)
            ridge_reg.fit(X_train, y_train)
```

```
            lasso_reg.fit(X_train, y_train)
            temp_lst2 = []
            temp_lst2.append(i)
            temp_lst2.append(a)
            temp_lst2.append(ridge_reg.score(X_train, y_train))
            temp_lst2.append(lasso_reg.score(X_train, y_train))
            temp_lst2.append(ridge_reg.score(X_test, y_test))
            temp_lst2.append(lasso_reg.score(X_test, y_test))
            temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst, columns=['k', 'alpha', 'Ridge Train Score',␣
 ↪'Lasso Train Score', 'Ridge Test Score', 'Lasso Test Score'])

temp_lst = []
for k in range(2, 10):
    for al in alphas:
        temp_lst2 = []
        temp_lst2.append(k)
        temp_lst2.append(al)
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['alpha'] == al)]['Ridge Train Score']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['alpha'] == al)]['Lasso Train Score']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['alpha'] == al)]['Ridge Test Score']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['alpha'] == al)]['Lasso Test Score']), decimals=4))
        temp_lst.append(temp_lst2)

ridge_lasso_reg_eval_df = pd.DataFrame(temp_lst, columns=['k', 'alpha', 'Ridge␣
 ↪Train Score', 'Lasso Train Score', 'Ridge Test Score', 'Lasso Test Score'])
ridge_lasso_reg_eval_df
```

```
100%|          | 8/8 [00:01<00:00,  5.52it/s]
```

[37]:

```
    k  alpha  Ridge Train Score  Lasso Train Score  Ridge Test Score  \
0   2   0.00               1.00               1.00              0.92
1   2   0.11               1.00               1.00              0.93
2   2   0.22               1.00               1.00              0.93
3   2   0.33               1.00               1.00              0.94
4   2   0.44               1.00               1.00              0.94
5   2   0.56               1.00               1.00              0.94
6   2   0.67               1.00               1.00              0.94
7   2   0.78               1.00               1.00              0.94
8   2   0.89               1.00               1.00              0.94
9   2   1.00               1.00               1.00              0.94
10  3   0.00               1.00               1.00              0.96
```

11	3	0.11	1.00	1.00	0.96
12	3	0.22	1.00	1.00	0.96
13	3	0.33	1.00	1.00	0.96
14	3	0.44	1.00	1.00	0.96
15	3	0.56	1.00	1.00	0.96
16	3	0.67	1.00	1.00	0.96
17	3	0.78	1.00	1.00	0.96
18	3	0.89	1.00	1.00	0.96
19	3	1.00	1.00	1.00	0.96
20	4	0.00	1.00	1.00	0.91
21	4	0.11	1.00	1.00	0.93
22	4	0.22	1.00	1.00	0.94
23	4	0.33	1.00	1.00	0.95
24	4	0.44	1.00	1.00	0.95
25	4	0.56	1.00	1.00	0.95
26	4	0.67	1.00	1.00	0.95
27	4	0.78	1.00	1.00	0.96
28	4	0.89	1.00	1.00	0.96
29	4	1.00	0.99	1.00	0.96
30	5	0.00	1.00	1.00	0.91
31	5	0.11	1.00	1.00	0.91
32	5	0.22	1.00	1.00	0.92
33	5	0.33	1.00	1.00	0.92
34	5	0.44	1.00	1.00	0.92
35	5	0.56	1.00	1.00	0.92
36	5	0.67	0.99	1.00	0.92
37	5	0.78	0.99	1.00	0.92
38	5	0.89	0.99	1.00	0.92
39	5	1.00	0.99	1.00	0.92
40	6	0.00	1.00	1.00	0.62
41	6	0.11	1.00	1.00	0.68
42	6	0.22	1.00	1.00	0.72
43	6	0.33	1.00	1.00	0.74
44	6	0.44	1.00	1.00	0.75
45	6	0.56	0.99	1.00	0.76
46	6	0.67	0.99	1.00	0.76
47	6	0.78	0.99	1.00	0.76
48	6	0.89	0.99	1.00	0.77
49	6	1.00	0.99	1.00	0.77
50	7	0.00	1.00	1.00	0.80
51	7	0.11	1.00	1.00	0.80
52	7	0.22	1.00	1.00	0.80
53	7	0.33	1.00	1.00	0.80
54	7	0.44	1.00	1.00	0.79
55	7	0.56	0.99	1.00	0.79
56	7	0.67	0.99	1.00	0.79
57	7	0.78	0.99	1.00	0.79

58	7	0.89	0.99	1.00	0.79
59	7	1.00	0.99	1.00	0.79
60	8	0.00	1.00	1.00	0.79
61	8	0.11	1.00	1.00	0.80
62	8	0.22	1.00	1.00	0.80
63	8	0.33	1.00	1.00	0.80
64	8	0.44	1.00	1.00	0.80
65	8	0.56	0.99	1.00	0.80
66	8	0.67	0.99	1.00	0.79
67	8	0.78	0.99	1.00	0.79
68	8	0.89	0.99	1.00	0.79
69	8	1.00	0.99	1.00	0.79
70	9	0.00	1.00	1.00	0.75
71	9	0.11	1.00	1.00	0.76
72	9	0.22	1.00	1.00	0.77
73	9	0.33	1.00	1.00	0.77
74	9	0.44	1.00	1.00	0.77
75	9	0.56	0.99	1.00	0.77
76	9	0.67	0.99	1.00	0.77
77	9	0.78	0.99	1.00	0.77
78	9	0.89	0.99	1.00	0.77
79	9	1.00	0.99	1.00	0.77

	Lasso Test Score
0	0.89
1	0.89
2	0.89
3	0.89
4	0.89
5	0.89
6	0.89
7	0.89
8	0.89
9	0.89
10	0.96
11	0.96
12	0.96
13	0.96
14	0.96
15	0.96
16	0.96
17	0.96
18	0.96
19	0.96
20	0.91
21	0.91
22	0.91

23	0.91
24	0.91
25	0.91
26	0.91
27	0.91
28	0.91
29	0.91
30	0.91
31	0.91
32	0.91
33	0.91
34	0.91
35	0.91
36	0.91
37	0.91
38	0.91
39	0.91
40	0.63
41	0.63
42	0.63
43	0.63
44	0.63
45	0.63
46	0.63
47	0.63
48	0.63
49	0.63
50	0.80
51	0.80
52	0.80
53	0.80
54	0.80
55	0.80
56	0.80
57	0.80
58	0.80
59	0.80
60	0.79
61	0.79
62	0.79
63	0.79
64	0.79
65	0.79
66	0.79
67	0.79
68	0.79
69	0.79

```
70                0.75
71                0.75
72                0.75
73                0.75
74                0.75
75                0.75
76                0.75
77                0.75
78                0.75
79                0.75
```

[38]:
```
ridge_lasso_reg_eval_df.nlargest(3, 'Ridge Test Score')
```

[38]:
```
    k  alpha  Ridge Train Score  Lasso Train Score  Ridge Test Score  \
11  3   0.11               1.00               1.00              0.96
12  3   0.22               1.00               1.00              0.96
10  3   0.00               1.00               1.00              0.96

    Lasso Test Score
11              0.96
12              0.96
10              0.96
```

[39]:
```
ridge_lasso_reg_eval_df.nsmallest(3, 'Ridge Test Score')
```

[39]:
```
    k  alpha  Ridge Train Score  Lasso Train Score  Ridge Test Score  \
40  6   0.00               1.00               1.00              0.62
41  6   0.11               1.00               1.00              0.68
42  6   0.22               1.00               1.00              0.72

    Lasso Test Score
40              0.63
41              0.63
42              0.63
```

as you can see in the tables above, the most accurate ridge and lasso models are 3 folded with with low alpha. also it's worthy to mention that the linear regression model in this dataset managed to achieve 96% accuracy also. based on that we train the final model with this combination.

[40]:
```
ridge = Ridge(alpha = 0.11)
lasso = Lasso(alpha = 0.11)
ridge.fit(ad_story_x, ad_story_y)
lasso.fit(ad_story_x, ad_story_y)
```

[40]:
```
Lasso(alpha=0.11)
```

Influencer

```
[41]: temp_lst = []
alphas = np.linspace(0, 1, 10)
for i in tqdm(range(2, 10)):
    kf = KFold(n_splits = i)
    for train_index, test_index in kf.split(influencer_x):
        X_train, X_test = influencer_x[train_index], influencer_x[test_index]
        y_train, y_test = influencer_y[train_index], influencer_y[test_index]
        for a in alphas:
            ridge_reg = Ridge(alpha=a)
            lasso_reg = Lasso(alpha=a)
            ridge_reg.fit(X_train, y_train)
            lasso_reg.fit(X_train, y_train)
            temp_lst2 = []
            temp_lst2.append(i)
            temp_lst2.append(a)
            temp_lst2.append(ridge_reg.score(X_train, y_train))
            temp_lst2.append(lasso_reg.score(X_train, y_train))
            temp_lst2.append(ridge_reg.score(X_test, y_test))
            temp_lst2.append(lasso_reg.score(X_test, y_test))
            temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst, columns=['k', 'alpha', 'Ridge Train Score',␣
↪'Lasso Train Score', 'Ridge Test Score', 'Lasso Test Score'])

temp_lst = []
for k in range(2, 10):
    for al in alphas:
        temp_lst2 = []
        temp_lst2.append(k)
        temp_lst2.append(al)
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['alpha'] == al)]['Ridge Train Score']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['alpha'] == al)]['Lasso Train Score']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['alpha'] == al)]['Ridge Test Score']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['alpha'] == al)]['Lasso Test Score']), decimals=4))
        temp_lst.append(temp_lst2)

ridge_lasso_reg_eval_df = pd.DataFrame(temp_lst, columns=['k', 'alpha', 'Ridge␣
↪Train Score', 'Lasso Train Score', 'Ridge Test Score', 'Lasso Test Score'])
ridge_lasso_reg_eval_df
```

```
100%|          | 8/8 [00:02<00:00,  3.56it/s]
```

[41]:

	k	alpha	Ridge Train Score	Lasso Train Score	Ridge Test Score	\
0	2	0.00	0.87	0.86	0.22	
1	2	0.11	0.87	0.86	0.28	
2	2	0.22	0.86	0.86	0.32	
3	2	0.33	0.86	0.86	0.35	
4	2	0.44	0.86	0.86	0.37	
5	2	0.56	0.86	0.86	0.39	
6	2	0.67	0.86	0.86	0.40	
7	2	0.78	0.86	0.86	0.41	
8	2	0.89	0.86	0.86	0.42	
9	2	1.00	0.86	0.86	0.43	
10	3	0.00	0.88	0.88	-0.55	
11	3	0.11	0.88	0.88	-0.23	
12	3	0.22	0.88	0.88	-0.14	
13	3	0.33	0.88	0.88	-0.07	
14	3	0.44	0.88	0.88	-0.03	
15	3	0.56	0.88	0.88	0.01	
16	3	0.67	0.88	0.88	0.03	
17	3	0.78	0.87	0.88	0.06	
18	3	0.89	0.87	0.88	0.07	
19	3	1.00	0.87	0.88	0.09	
20	4	0.00	0.85	0.85	-0.17	
21	4	0.11	0.85	0.85	-0.10	
22	4	0.22	0.85	0.85	-0.06	
23	4	0.33	0.85	0.85	-0.03	
24	4	0.44	0.85	0.85	-0.00	
25	4	0.56	0.85	0.85	0.01	
26	4	0.67	0.85	0.85	0.03	
27	4	0.78	0.85	0.85	0.04	
28	4	0.89	0.85	0.85	0.05	
29	4	1.00	0.85	0.85	0.06	
30	5	0.00	0.85	0.85	-0.30	
31	5	0.11	0.85	0.85	-0.23	
32	5	0.22	0.85	0.85	-0.19	
33	5	0.33	0.85	0.85	-0.15	
34	5	0.44	0.85	0.85	-0.12	
35	5	0.56	0.85	0.85	-0.10	
36	5	0.67	0.85	0.85	-0.08	
37	5	0.78	0.85	0.85	-0.06	
38	5	0.89	0.85	0.85	-0.05	
39	5	1.00	0.85	0.85	-0.04	
40	6	0.00	0.85	0.85	-0.16	
41	6	0.11	0.85	0.85	-0.07	
42	6	0.22	0.85	0.85	-0.03	
43	6	0.33	0.85	0.85	-0.00	
44	6	0.44	0.85	0.85	0.02	
45	6	0.56	0.85	0.85	0.04	

46	6	0.67	0.85	0.85	0.06
47	6	0.78	0.85	0.85	0.07
48	6	0.89	0.85	0.85	0.08
49	6	1.00	0.85	0.85	0.09
50	7	0.00	0.84	0.84	-1.00
51	7	0.11	0.84	0.84	-0.97
52	7	0.22	0.84	0.84	-0.94
53	7	0.33	0.84	0.84	-0.93
54	7	0.44	0.84	0.84	-0.92
55	7	0.56	0.84	0.84	-0.91
56	7	0.67	0.84	0.84	-0.90
57	7	0.78	0.84	0.84	-0.89
58	7	0.89	0.84	0.84	-0.89
59	7	1.00	0.84	0.84	-0.89
60	8	0.00	0.84	0.84	-0.59
61	8	0.11	0.85	0.84	-0.53
62	8	0.22	0.84	0.84	-0.49
63	8	0.33	0.84	0.84	-0.46
64	8	0.44	0.84	0.84	-0.44
65	8	0.56	0.84	0.84	-0.42
66	8	0.67	0.84	0.84	-0.40
67	8	0.78	0.84	0.84	-0.39
68	8	0.89	0.84	0.84	-0.38
69	8	1.00	0.84	0.84	-0.37
70	9	0.00	0.84	0.84	-1.93
71	9	0.11	0.84	0.84	-1.89
72	9	0.22	0.84	0.84	-1.86
73	9	0.33	0.84	0.84	-1.85
74	9	0.44	0.84	0.84	-1.83
75	9	0.56	0.84	0.84	-1.82
76	9	0.67	0.84	0.84	-1.81
77	9	0.78	0.84	0.84	-1.80
78	9	0.89	0.84	0.84	-1.80
79	9	1.00	0.84	0.84	-1.79

	Lasso Test Score
0	0.40
1	0.40
2	0.40
3	0.40
4	0.40
5	0.40
6	0.40
7	0.40
8	0.40
9	0.40
10	-0.26

11	-0.26
12	-0.26
13	-0.26
14	-0.26
15	-0.26
16	-0.26
17	-0.26
18	-0.26
19	-0.26
20	-0.08
21	-0.08
22	-0.08
23	-0.08
24	-0.08
25	-0.08
26	-0.08
27	-0.08
28	-0.08
29	-0.08
30	-0.25
31	-0.25
32	-0.25
33	-0.25
34	-0.25
35	-0.25
36	-0.25
37	-0.25
38	-0.25
39	-0.25
40	-0.04
41	-0.04
42	-0.04
43	-0.04
44	-0.04
45	-0.04
46	-0.04
47	-0.04
48	-0.04
49	-0.04
50	-0.92
51	-0.92
52	-0.92
53	-0.92
54	-0.92
55	-0.92
56	-0.92
57	-0.92

```
58             -0.92
59             -0.92
60             -0.61
61             -0.61
62             -0.61
63             -0.61
64             -0.61
65             -0.61
66             -0.61
67             -0.61
68             -0.61
69             -0.61
70             -1.90
71             -1.90
72             -1.90
73             -1.90
74             -1.90
75             -1.90
76             -1.90
77             -1.90
78             -1.90
79             -1.90
```

[42]:
```
ridge_lasso_reg_eval_df.nlargest(3, 'Ridge Test Score')
```

[42]:
```
   k  alpha  Ridge Train Score  Lasso Train Score  Ridge Test Score  \
9  2   1.00               0.86               0.86              0.43
8  2   0.89               0.86               0.86              0.42
7  2   0.78               0.86               0.86              0.41

   Lasso Test Score
9              0.40
8              0.40
7              0.40
```

[43]:
```
ridge_lasso_reg_eval_df.nsmallest(3, 'Ridge Test Score')
```

[43]:
```
    k  alpha  Ridge Train Score  Lasso Train Score  Ridge Test Score  \
70  9   0.00               0.84               0.84             -1.93
71  9   0.11               0.84               0.84             -1.89
72  9   0.22               0.84               0.84             -1.86

    Lasso Test Score
70             -1.90
71             -1.90
72             -1.90
```

as you can see, like linear regression, these algorithms are not very good fit for this dataset. as you can see the most accurate ridge model managed to get 43% accuracy and most accurate lasso model managed to get jsut 40% accuracy which is not very good. this fact indicates that the behavior of data in this dataset is not linear.

we anticipate the same concept is applied for leaders_post dataset.

Leaders Posts

[44]:
```
temp_lst = []
alphas = np.linspace(0, 1, 10)
for i in tqdm(range(2, 10)):
    kf = KFold(n_splits = i)
    for train_index, test_index in kf.split(leaders_post_x):
        X_train, X_test = leaders_post_x[train_index],␣
↪leaders_post_x[test_index]
        y_train, y_test = leaders_post_y[train_index],␣
↪leaders_post_y[test_index]
        for a in alphas:
            ridge_reg = Ridge(alpha=a)
            lasso_reg = Lasso(alpha=a)
            ridge_reg.fit(X_train, y_train)
            lasso_reg.fit(X_train, y_train)
            temp_lst2 = []
            temp_lst2.append(i)
            temp_lst2.append(a)
            temp_lst2.append(ridge_reg.score(X_train, y_train))
            temp_lst2.append(lasso_reg.score(X_train, y_train))
            temp_lst2.append(ridge_reg.score(X_test, y_test))
            temp_lst2.append(lasso_reg.score(X_test, y_test))
            temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst, columns=['k', 'alpha', 'Ridge Train Score',␣
↪'Lasso Train Score', 'Ridge Test Score', 'Lasso Test Score'])

temp_lst = []
for k in range(2, 10):
    for al in alphas:
        temp_lst2 = []
        temp_lst2.append(k)
        temp_lst2.append(al)
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['alpha'] == al)]['Ridge Train Score']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['alpha'] == al)]['Lasso Train Score']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['alpha'] == al)]['Ridge Test Score']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['alpha'] == al)]['Lasso Test Score']), decimals=4))
```

```
        temp_lst.append(temp_lst2)

ridge_lasso_reg_eval_df = pd.DataFrame(temp_lst, columns=['k', 'alpha', 'Ridge␣
 ↪Train Score', 'Lasso Train Score', 'Ridge Test Score', 'Lasso Test Score'])
ridge_lasso_reg_eval_df
```

100%| | 8/8 [00:01<00:00, 6.41it/s]

[44]:

	k	alpha	Ridge Train Score	Lasso Train Score	Ridge Test Score	\
0	2	0.00	1.00	1.00	-119.27	
1	2	0.11	1.00	1.00	-119.27	
2	2	0.22	1.00	1.00	-119.27	
3	2	0.33	1.00	1.00	-119.27	
4	2	0.44	1.00	1.00	-119.27	
5	2	0.56	1.00	1.00	-119.27	
6	2	0.67	1.00	1.00	-119.27	
7	2	0.78	1.00	1.00	-119.27	
8	2	0.89	1.00	1.00	-119.27	
9	2	1.00	1.00	1.00	-119.27	
10	3	0.00	1.00	1.00	-51.77	
11	3	0.11	1.00	1.00	-51.77	
12	3	0.22	1.00	1.00	-51.77	
13	3	0.33	1.00	1.00	-51.77	
14	3	0.44	1.00	1.00	-51.77	
15	3	0.56	1.00	1.00	-51.77	
16	3	0.67	1.00	1.00	-51.77	
17	3	0.78	1.00	1.00	-51.77	
18	3	0.89	1.00	1.00	-51.77	
19	3	1.00	1.00	1.00	-51.77	
20	4	0.00	1.00	1.00	-200198.36	
21	4	0.11	1.00	1.00	-200197.32	
22	4	0.22	1.00	1.00	-200196.28	
23	4	0.33	1.00	1.00	-200195.24	
24	4	0.44	1.00	1.00	-200194.20	
25	4	0.56	1.00	1.00	-200193.16	
26	4	0.67	1.00	1.00	-200192.12	
27	4	0.78	1.00	1.00	-200191.08	
28	4	0.89	1.00	1.00	-200190.04	
29	4	1.00	1.00	1.00	-200189.00	
30	5	0.00	1.00	1.00	-1440.19	
31	5	0.11	1.00	1.00	-1440.18	
32	5	0.22	1.00	1.00	-1440.17	
33	5	0.33	1.00	1.00	-1440.16	
34	5	0.44	1.00	1.00	-1440.14	
35	5	0.56	1.00	1.00	-1440.13	
36	5	0.67	1.00	1.00	-1440.12	
37	5	0.78	1.00	1.00	-1440.11	

38	5	0.89	1.00	1.00	-1440.10
39	5	1.00	1.00	1.00	-1440.09
40	6	0.00	1.00	1.00	-1770.67
41	6	0.11	1.00	1.00	-1770.66
42	6	0.22	1.00	1.00	-1770.64
43	6	0.33	1.00	1.00	-1770.63
44	6	0.44	1.00	1.00	-1770.61
45	6	0.56	1.00	1.00	-1770.60
46	6	0.67	1.00	1.00	-1770.59
47	6	0.78	1.00	1.00	-1770.57
48	6	0.89	1.00	1.00	-1770.56
49	6	1.00	1.00	1.00	-1770.54
50	7	0.00	1.00	1.00	-2655.51
51	7	0.11	1.00	1.00	-2655.49
52	7	0.22	1.00	1.00	-2655.47
53	7	0.33	1.00	1.00	-2655.45
54	7	0.44	1.00	1.00	-2655.42
55	7	0.56	1.00	1.00	-2655.40
56	7	0.67	1.00	1.00	-2655.38
57	7	0.78	1.00	1.00	-2655.36
58	7	0.89	1.00	1.00	-2655.33
59	7	1.00	1.00	1.00	-2655.31
60	8	0.00	1.00	1.00	-52.77
61	8	0.11	1.00	1.00	-52.77
62	8	0.22	1.00	1.00	-52.77
63	8	0.33	1.00	1.00	-52.77
64	8	0.44	1.00	1.00	-52.77
65	8	0.56	1.00	1.00	-52.77
66	8	0.67	1.00	1.00	-52.77
67	8	0.78	1.00	1.00	-52.77
68	8	0.89	1.00	1.00	-52.77
69	8	1.00	1.00	1.00	-52.77
70	9	0.00	1.00	1.00	nan
71	9	0.11	1.00	1.00	nan
72	9	0.22	1.00	1.00	nan
73	9	0.33	1.00	1.00	nan
74	9	0.44	1.00	1.00	nan
75	9	0.56	1.00	1.00	nan
76	9	0.67	1.00	1.00	nan
77	9	0.78	1.00	1.00	nan
78	9	0.89	1.00	1.00	nan
79	9	1.00	1.00	1.00	nan

	Lasso Test Score
0	-3.06
1	-3.06
2	-3.06

3	-3.06
4	-3.06
5	-3.06
6	-3.06
7	-3.06
8	-3.06
9	-3.06
10	-294.25
11	-294.25
12	-294.25
13	-294.25
14	-294.25
15	-294.25
16	-294.25
17	-294.25
18	-294.25
19	-294.25
20	-1323.87
21	-1323.87
22	-1323.87
23	-1323.87
24	-1323.87
25	-1323.87
26	-1323.87
27	-1323.87
28	-1323.87
29	-1323.87
30	-975.02
31	-975.02
32	-975.02
33	-975.02
34	-975.02
35	-975.02
36	-975.02
37	-975.02
38	-975.02
39	-975.02
40	-1296.63
41	-1296.63
42	-1296.63
43	-1296.63
44	-1296.63
45	-1296.63
46	-1296.63
47	-1296.63
48	-1296.63
49	-1296.63

```
50          -1944.31
51          -1944.31
52          -1944.31
53          -1944.31
54          -1944.31
55          -1944.31
56          -1944.31
57          -1944.31
58          -1944.31
59          -1944.31
60             -9.15
61             -9.15
62             -9.15
63             -9.15
64             -9.15
65             -9.15
66             -9.15
67             -9.15
68             -9.15
69             -9.15
70               nan
71               nan
72               nan
73               nan
74               nan
75               nan
76               nan
77               nan
78               nan
79               nan
```

[45]:
```
ridge_lasso_reg_eval_df.nlargest(3, 'Ridge Test Score')
```

[45]:
```
    k  alpha  Ridge Train Score  Lasso Train Score  Ridge Test Score  \
19  3   1.00               1.00               1.00            -51.77
14  3   0.44               1.00               1.00            -51.77
15  3   0.56               1.00               1.00            -51.77

    Lasso Test Score
19           -294.25
14           -294.25
15           -294.25
```

[46]:
```
ridge_lasso_reg_eval_df.nsmallest(3, 'Ridge Test Score')
```

[46]:
```
    k  alpha  Ridge Train Score  Lasso Train Score  Ridge Test Score  \
20  4   0.00               1.00               1.00        -200198.36
```

```
21  4   0.11                1.00                1.00         -200197.32
22  4   0.22                1.00                1.00         -200196.28

    Lasso Test Score
20          -1323.87
21          -1323.87
22          -1323.87
```

as you can see in the tables above and as it was anticipated the performance of lasso and ridge algorithms in leaders_post dataset are significantly bad and linear models are not a good algorithms to fit to these data.

1.1.4 Support Vector Machine Regressor

Advertising Posts

[47]:
```python
from sklearn.svm import SVR
```

[55]:
```python
c_lst = [1, .5]
kernel_lst = ['linear', 'rbf']
```

[57]:
```python
temp_lst = []
for i in tqdm(range(2, 5)):
    kf = KFold(n_splits = i)
    for train_index, test_index in kf.split(ad_post_x):
        X_train, X_test = ad_post_x[train_index], ad_post_x[test_index]
        y_train, y_test = ad_post_y[train_index], ad_post_y[test_index]
        for c in c_lst:
            for kernel_type in kernel_lst:
                reg_svm = SVR(C=c, kernel=kernel_type)
                reg_svm.fit(X_train, y_train)
                temp_lst2 = []
                temp_lst2.append(i)
                temp_lst2.append(c)
                temp_lst2.append(kernel_type)
                temp_lst2.append(reg_svm.score(X_train, y_train))
                temp_lst2.append(reg_svm.score(X_test, y_test))
                temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst,
                       columns=['k', 'c', 'kernel', 'Train Score', 'Test␣
↪Score'])
temp_lst = []
for k in range(2, 5):
    for c in c_lst:
        for kernel_type in kernel_lst:
            temp_lst2 = []
            temp_lst2.append(k)
            temp_lst2.append(c)
```

```
            temp_lst2.append(kernel_type)
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['c'] == c) & (temp_df['kernel'] == kernel_type)]['Train Score']),␣
↪decimals=4))
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['c'] == c) & (temp_df['kernel'] == kernel_type)]['Test Score']),␣
↪decimals=4))
            temp_lst.append(temp_lst2)

reg_svm_eval_df = pd.DataFrame(temp_lst,
                               columns=['k', 'c', 'kernel', 'Train Score', 'Test␣
↪Score'])
reg_svm_eval_df
```

100%| | 3/3 [04:21<00:00, 87.19s/it]

[57]:

	k	c	kernel	Train Score	Test Score
0	2	1.00	linear	0.86	0.61
1	2	1.00	rbf	-0.03	-0.15
2	2	0.50	linear	0.86	0.59
3	2	0.50	rbf	-0.03	-0.15
4	3	1.00	linear	0.86	0.74
5	3	1.00	rbf	-0.08	-0.22
6	3	0.50	linear	0.87	0.70
7	3	0.50	rbf	-0.08	-0.22
8	4	1.00	linear	0.84	0.54
9	4	1.00	rbf	-0.11	-0.42
10	4	0.50	linear	0.84	0.53
11	4	0.50	rbf	-0.11	-0.42

[58]:

```
reg_svm_eval_df.nlargest(3, 'Test Score')
```

[58]:

	k	c	kernel	Train Score	Test Score
4	3	1.00	linear	0.86	0.74
6	3	0.50	linear	0.87	0.70
0	2	1.00	linear	0.86	0.61

[59]:

```
reg_svm_eval_df.nsmallest(3, 'Test Score')
```

[59]:

	k	c	kernel	Train Score	Test Score
9	4	1.00	rbf	-0.11	-0.42
11	4	0.50	rbf	-0.11	-0.42
5	3	1.00	rbf	-0.08	-0.22

as you can see the svm regressor is not better performing than linear regression on these dataset, but it take so much longer time to train since the calculations for svm is much more complicated and more costly than linear regression.

Advertising Stories

[60]:
```
temp_lst = []
for i in tqdm(range(2, 5)):
    kf = KFold(n_splits = i)
    for train_index, test_index in kf.split(ad_story_x):
        X_train, X_test = ad_story_x[train_index], ad_story_x[test_index]
        y_train, y_test = ad_story_y[train_index], ad_story_y[test_index]
        for c in c_lst:
            for kernel_type in kernel_lst:
                reg_svm = SVR(C=c, kernel=kernel_type)
                reg_svm.fit(X_train, y_train)
                temp_lst2 = []
                temp_lst2.append(i)
                temp_lst2.append(c)
                temp_lst2.append(kernel_type)
                temp_lst2.append(reg_svm.score(X_train, y_train))
                temp_lst2.append(reg_svm.score(X_test, y_test))
                temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst,
                       columns=['k', 'c', 'kernel', 'Train Score', 'Test␣
 ↪Score'])
temp_lst = []
for k in range(2, 5):
    for c in c_lst:
        for kernel_type in kernel_lst:
            temp_lst2 = []
            temp_lst2.append(k)
            temp_lst2.append(c)
            temp_lst2.append(kernel_type)
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['c'] == c) & (temp_df['kernel'] == kernel_type)]['Train Score']),␣
 ↪decimals=4))
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['c'] == c) & (temp_df['kernel'] == kernel_type)]['Test Score']),␣
 ↪decimals=4))
            temp_lst.append(temp_lst2)

reg_svm_eval_df = pd.DataFrame(temp_lst,
                               columns=['k', 'c', 'kernel', 'Train Score', 'Test␣
 ↪Score'])
reg_svm_eval_df
```

```
100%|          | 3/3 [03:46<00:00, 75.59s/it]
```

[60]:

	k	c	kernel	Train Score	Test Score
0	2	1.00	linear	0.95	0.90

```
1    2 1.00     rbf        -0.14        -0.16
2    2 0.50  linear         0.97         0.91
3    2 0.50     rbf        -0.14        -0.16
4    3 1.00  linear         0.92         0.82
5    3 1.00     rbf        -0.11        -0.23
6    3 0.50  linear         0.97         0.94
7    3 0.50     rbf        -0.11        -0.23
8    4 1.00  linear         0.93         0.59
9    4 1.00     rbf        -0.11        -1.93
10   4 0.50  linear         0.97         0.89
11   4 0.50     rbf        -0.11        -1.93
```

[61]:
```
reg_svm_eval_df.nlargest(3, 'Test Score')
```

[61]:

	k	c	kernel	Train Score	Test Score
6	3	0.50	linear	0.97	0.94
2	2	0.50	linear	0.97	0.91
0	2	1.00	linear	0.95	0.90

[62]:
```
reg_svm_eval_df.nsmallest(3, 'Test Score')
```

[62]:

	k	c	kernel	Train Score	Test Score
9	4	1.00	rbf	-0.11	-1.93
11	4	0.50	rbf	-0.11	-1.93
5	3	1.00	rbf	-0.11	-0.23

as you can see although the performance of svm regressor in advertising stories are fine, but its performance are not better than linear regression model. we will anticipate that the performance of svm regressor in influencer and leaders_post dataset will be terrible.

Influencers

[63]:
```
temp_lst = []
for i in tqdm(range(2, 5)):
    kf = KFold(n_splits = i)
    for train_index, test_index in kf.split(influencer_x):
        X_train, X_test = influencer_x[train_index], influencer_x[test_index]
        y_train, y_test = influencer_y[train_index], influencer_y[test_index]
        for c in c_lst:
            for kernel_type in kernel_lst:
                reg_svm = SVR(C=c, kernel=kernel_type)
                reg_svm.fit(X_train, y_train)
                temp_lst2 = []
                temp_lst2.append(i)
                temp_lst2.append(c)
                temp_lst2.append(kernel_type)
                temp_lst2.append(reg_svm.score(X_train, y_train))
                temp_lst2.append(reg_svm.score(X_test, y_test))
                temp_lst.append(temp_lst2)
```

```
temp_df = pd.DataFrame(temp_lst,
                       columns=['k', 'c', 'kernel', 'Train Score', 'Test␣
↪Score'])
temp_lst = []
for k in range(2, 5):
    for c in c_lst:
        for kernel_type in kernel_lst:
            temp_lst2 = []
            temp_lst2.append(k)
            temp_lst2.append(c)
            temp_lst2.append(kernel_type)
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['c'] == c) & (temp_df['kernel'] == kernel_type)]['Train Score']),␣
↪decimals=4))
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['c'] == c) & (temp_df['kernel'] == kernel_type)]['Test Score']),␣
↪decimals=4))
            temp_lst.append(temp_lst2)

reg_svm_eval_df = pd.DataFrame(temp_lst,
                               columns=['k', 'c', 'kernel', 'Train Score', 'Test␣
↪Score'])
reg_svm_eval_df
```

```
100%|          | 3/3 [02:01<00:00, 40.57s/it]
```

[63]:

	k	c	kernel	Train Score	Test Score
0	2	1.00	linear	0.72	0.60
1	2	1.00	rbf	-0.10	-0.54
2	2	0.50	linear	0.72	0.65
3	2	0.50	rbf	-0.10	-0.54
4	3	1.00	linear	0.78	0.30
5	3	1.00	rbf	-0.06	-0.53
6	3	0.50	linear	0.78	0.32
7	3	0.50	rbf	-0.06	-0.53
8	4	1.00	linear	0.77	-0.57
9	4	1.00	rbf	-0.10	-1.02
10	4	0.50	linear	0.77	-0.00
11	4	0.50	rbf	-0.10	-1.02

[64]:

```
reg_svm_eval_df.nlargest(3, 'Test Score')
```

[64]:

	k	c	kernel	Train Score	Test Score
2	2	0.50	linear	0.72	0.65
0	2	1.00	linear	0.72	0.60
6	3	0.50	linear	0.78	0.32

[65]:
```
reg_svm_eval_df.nsmallest(3, 'Test Score')
```

[65]:

	k	c	kernel	Train Score	Test Score
9	4	1.00	rbf	-0.10	-1.02
11	4	0.50	rbf	-0.10	-1.02
8	4	1.00	linear	0.77	-0.57

surprisingly svm performed much better than previous regression algorithms in influencers dataset. although its score and accuracy are not particularly good, but it performed much better than the others on this dataset.

Leader Posts

[66]:
```
temp_lst = []
for i in tqdm(range(2, 5)):
    kf = KFold(n_splits = i)
    for train_index, test_index in kf.split(leaders_post_x):
        X_train, X_test = leaders_post_x[train_index],␣
↪leaders_post_x[test_index]
        y_train, y_test = leaders_post_y[train_index],␣
↪leaders_post_y[test_index]
        for c in c_lst:
            for kernel_type in kernel_lst:
                reg_svm = SVR(C=c, kernel=kernel_type)
                reg_svm.fit(X_train, y_train)
                temp_lst2 = []
                temp_lst2.append(i)
                temp_lst2.append(c)
                temp_lst2.append(kernel_type)
                temp_lst2.append(reg_svm.score(X_train, y_train))
                temp_lst2.append(reg_svm.score(X_test, y_test))
                temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst,
                       columns=['k', 'c', 'kernel', 'Train Score', 'Test␣
↪Score'])
temp_lst = []
for k in range(2, 5):
    for c in c_lst:
        for kernel_type in kernel_lst:
            temp_lst2 = []
            temp_lst2.append(k)
            temp_lst2.append(c)
            temp_lst2.append(kernel_type)
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['c'] == c) & (temp_df['kernel'] == kernel_type)]['Train Score']),␣
↪decimals=4))
```

```
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['c'] == c) & (temp_df['kernel'] == kernel_type)]['Test Score']),␣
↪decimals=4))
            temp_lst.append(temp_lst2)

reg_svm_eval_df = pd.DataFrame(temp_lst,
                              columns=['k', 'c', 'kernel', 'Train Score', 'Test␣
↪Score'])
reg_svm_eval_df
```

```
100%|          | 3/3 [00:00<00:00, 13.16it/s]
```

[66]:

	k	c	kernel	Train Score	Test Score
0	2	1.00	linear	1.00	-119.27
1	2	1.00	rbf	-0.21	-0.42
2	2	0.50	linear	0.95	-29.20
3	2	0.50	rbf	-0.21	-0.42
4	3	1.00	linear	0.81	0.07
5	3	1.00	rbf	-0.11	-0.32
6	3	0.50	linear	0.80	0.16
7	3	0.50	rbf	-0.11	-0.32
8	4	1.00	linear	0.80	-103.11
9	4	1.00	rbf	-0.18	-145.12
10	4	0.50	linear	0.80	-70.90
11	4	0.50	rbf	-0.18	-145.12

[67]:

```
reg_svm_eval_df.nlargest(3, 'Test Score')
```

[67]:

	k	c	kernel	Train Score	Test Score
6	3	0.50	linear	0.80	0.16
4	3	1.00	linear	0.81	0.07
5	3	1.00	rbf	-0.11	-0.32

like pervious regression algorithms, svm performed very bad on leaders post dataset and underfitted significantly, as you can see in table above, although it managed to achieve 81% in training dataset, but its accuracy on test dataset was 7% which is terrible.

in the next note book we will check the non-linear approaches for regression in these datasets.

---

2 Notebook by Ramin F. - @simplyramin

[ ]:

```
```