Modeling-Classification

June 2, 2021

1 Modeling

1.1 Classification

in this notebook we are going to apporach the classification problem, our available datasets consits of two classification problems, binary and multiclass. although some machine learning algorithms are capable of both, there are some which can only do binary or multiclass classification. so, if a specific algorithm is not performed for a dataset, its because that datasets classification problem couldn't be done with that specific algorithm.

---

[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")
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_notebook, tqdm
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn import metrics
import itertools
import seaborn as sns
```

[2]:
```
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.
```

[3]:
```
#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()
```

[5]:
```
ad_post_y = np.asarray(ad_post_dummy[['benefit']])
ad_post_x = np.asarray(ad_post_dummy[['follower', 'view', 'cost', 'field_art &␣
↪culture', 'field_fact', 'field_video', 'field_women']])

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

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

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

1.1.1 Logistic Regression (Both)

Advertising Post

[6]:
```
from sklearn.linear_model import LogisticRegression
from sklearn import preprocessing
```

Normalizing independent variables:

[7]:
```
ad_post_x = preprocessing.StandardScaler().fit(ad_post_x).transform(ad_post_x)
ad_story_x = preprocessing.StandardScaler().fit(ad_story_x).
↪transform(ad_story_x)
influencer_x = preprocessing.StandardScaler().fit(influencer_x).
↪transform(influencer_x)
leaders_post_x = preprocessing.StandardScaler().fit(leaders_post_x).
↪transform(leaders_post_x)
```

[8]:
```
c_lst = [1, .5, .25, .1, .05, .025, .01, .005, .0025, .001]
```

[9]:

```
temp_lst = []
for i in range(2, 6):
    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:
            clf_lr = LogisticRegression(penalty='l2', C=c, solver='liblinear')
            clf_lr.fit(X_train, y_train)
            y_hat = clf_lr.predict(X_test)
            y_hat_prob = clf_lr.predict_proba(X_test)
            temp_lst2 = []
            temp_lst2.append(i)
            temp_lst2.append(c)
            temp_lst2.append(metrics.accuracy_score(y_train, clf_lr.
 ↪predict(X_train)))
            temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
            temp_lst2.append(metrics.f1_score(y_test, y_hat))
            temp_lst2.append(metrics.jaccard_score(y_test, y_hat))
            temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst,
                       columns=['k', 'c', 'Train-set Accuracy', 'Test-set␣
 ↪Accuracy', 'F1 Score', 'Jaccard Score'])
temp_lst = []
for k in range(2, 6):
    for c in c_lst:
        temp_lst2 = []
        temp_lst2.append(k)
        temp_lst2.append(c)
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['c'] == c)]['Train-set Accuracy']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['c'] == c)]['Test-set Accuracy']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['c'] == c)]['F1 Score']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['c'] == c)]['Jaccard Score']), decimals=4))
        temp_lst.append(temp_lst2)

clf_lr_eval_df = pd.DataFrame(temp_lst,
                              columns=['k', 'c', 'Train-set Accuracy',␣
 ↪'Test-set Accuracy', 'F1 Score', 'Jaccard Score'])
clf_lr_eval_df
```

[9]:

	k	c	Train-set Accuracy	Test-set Accuracy	F1 Score	Jaccard Score
0	2	1.00	0.85	0.63	0.70	0.54
1	2	0.50	0.81	0.59	0.64	0.47
2	2	0.25	0.81	0.59	0.64	0.47
3	2	0.10	0.81	0.52	0.58	0.41
4	2	0.05	0.81	0.52	0.58	0.41
5	2	0.03	0.81	0.56	0.60	0.43
6	2	0.01	0.81	0.56	0.60	0.43
7	2	0.01	0.81	0.56	0.60	0.43
8	2	0.00	0.81	0.56	0.60	0.43
9	2	0.00	0.81	0.56	0.60	0.43
10	3	1.00	0.87	0.74	0.72	0.66
11	3	0.50	0.85	0.74	0.72	0.66
12	3	0.25	0.80	0.74	0.72	0.66
13	3	0.10	0.78	0.78	0.73	0.67
14	3	0.05	0.76	0.78	0.72	0.63
15	3	0.03	0.74	0.74	0.69	0.57
16	3	0.01	0.70	0.70	0.64	0.52
17	3	0.01	0.70	0.70	0.64	0.52
18	3	0.00	0.70	0.70	0.64	0.52
19	3	0.00	0.70	0.70	0.64	0.52
20	4	1.00	0.86	0.69	0.71	0.61
21	4	0.50	0.85	0.69	0.71	0.61
22	4	0.25	0.83	0.69	0.71	0.61
23	4	0.10	0.80	0.65	0.69	0.57
24	4	0.05	0.76	0.62	0.64	0.51
25	4	0.03	0.76	0.67	0.62	0.48
26	4	0.01	0.74	0.67	0.62	0.48
27	4	0.01	0.75	0.67	0.62	0.48
28	4	0.00	0.75	0.67	0.62	0.48
29	4	0.00	0.75	0.67	0.62	0.48
30	5	1.00	0.87	0.69	0.70	0.59
31	5	0.50	0.81	0.69	0.70	0.59
32	5	0.25	0.80	0.66	0.67	0.55
33	5	0.10	0.80	0.63	0.63	0.51
34	5	0.05	0.81	0.63	0.63	0.51
35	5	0.03	0.82	0.67	0.65	0.53
36	5	0.01	0.77	0.67	0.65	0.53
37	5	0.01	0.75	0.67	0.65	0.53
38	5	0.00	0.75	0.67	0.65	0.53
39	5	0.00	0.74	0.67	0.65	0.53

[10]:

```
clf_lr_eval_df.nlargest(3, 'Test-set Accuracy')
```

[10]:

	k	c	Train-set Accuracy	Test-set Accuracy	F1 Score	Jaccard Score
13	3	0.10	0.78	0.78	0.73	0.67
14	3	0.05	0.76	0.78	0.72	0.63

```
10  3 1.00                0.87               0.74       0.72            0.66
```

[11]:

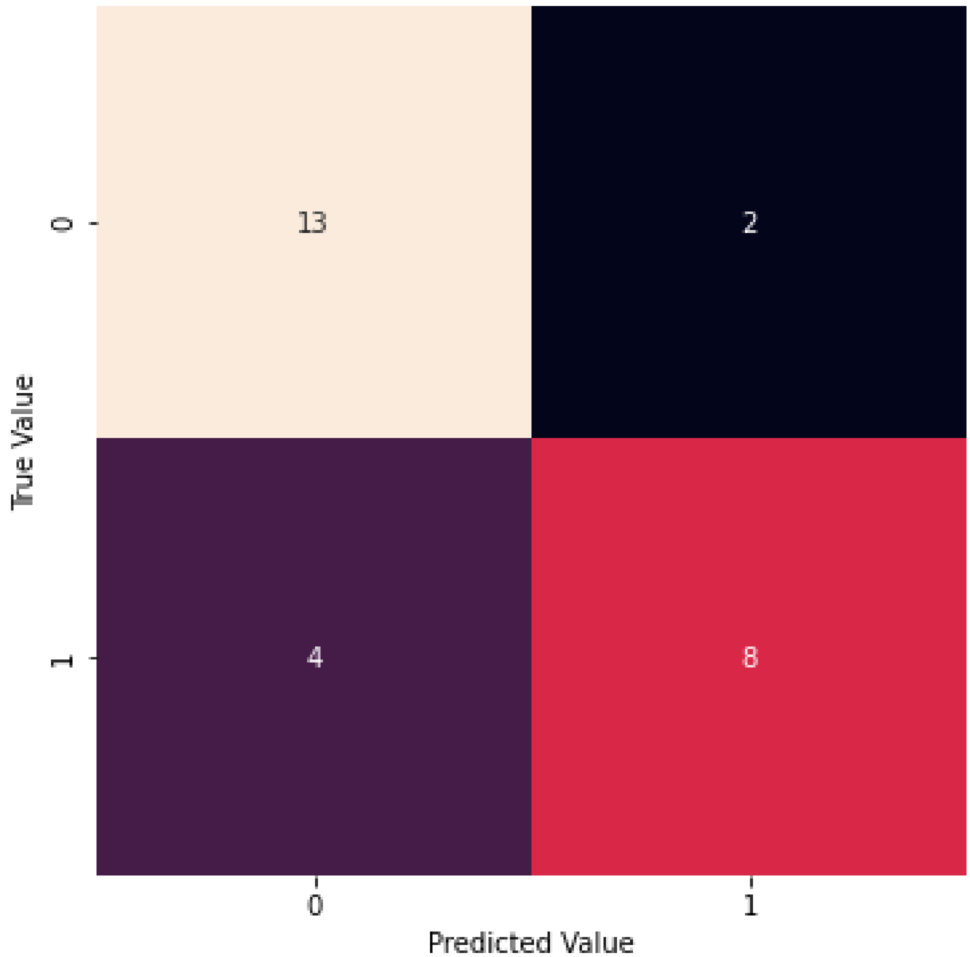
```python
kf = KFold(n_splits = 3)
temp_lst = []
clf_lr = LogisticRegression(penalty='l2', C=.10, solver='liblinear')
for train_index, test_index in kf.split(ad_post_x):
    X_train, X_test = ad_post_x[train_index], ad_post_x[test_index]
    y_train, y_test = ad_post_y[train_index], ad_post_y[test_index]
    clf_lr.fit(X_train, y_train)
    y_hat = clf_lr.predict(X_test)
    y_hat_prob = clf_lr.predict_proba(X_test)
    temp_lst2 = []
    temp_lst2.append(metrics.accuracy_score(y_train, clf_lr.predict(X_train)))
    temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
    temp_lst2.append(metrics.f1_score(y_test, y_hat))
    temp_lst2.append(metrics.jaccard_score(y_test, y_hat))
    temp_lst2.append(metrics.log_loss(y_test, y_hat_prob))
    temp_lst2.append(y_test)
    temp_lst2.append(y_hat)
    temp_lst2.append(X_test)
    temp_lst.append(temp_lst2)
```

[12]:

```python
temp_lst_ytest, temp_lst_yhat, temp_lst_xtest = [], [], []
for row in temp_lst:
    for i in row[5]:
        temp_lst_ytest.append(i)
    for j in row[6]:
        temp_lst_yhat.append(j)
    for k in row[7]:
        temp_lst_xtest.append(k)

cnf_ytest = np.array(temp_lst_ytest)
cnf_yhat = np.array(temp_lst_yhat)
cnf_xtest = np.array(temp_lst_xtest)
print(metrics.classification_report(cnf_ytest, cnf_yhat))
```

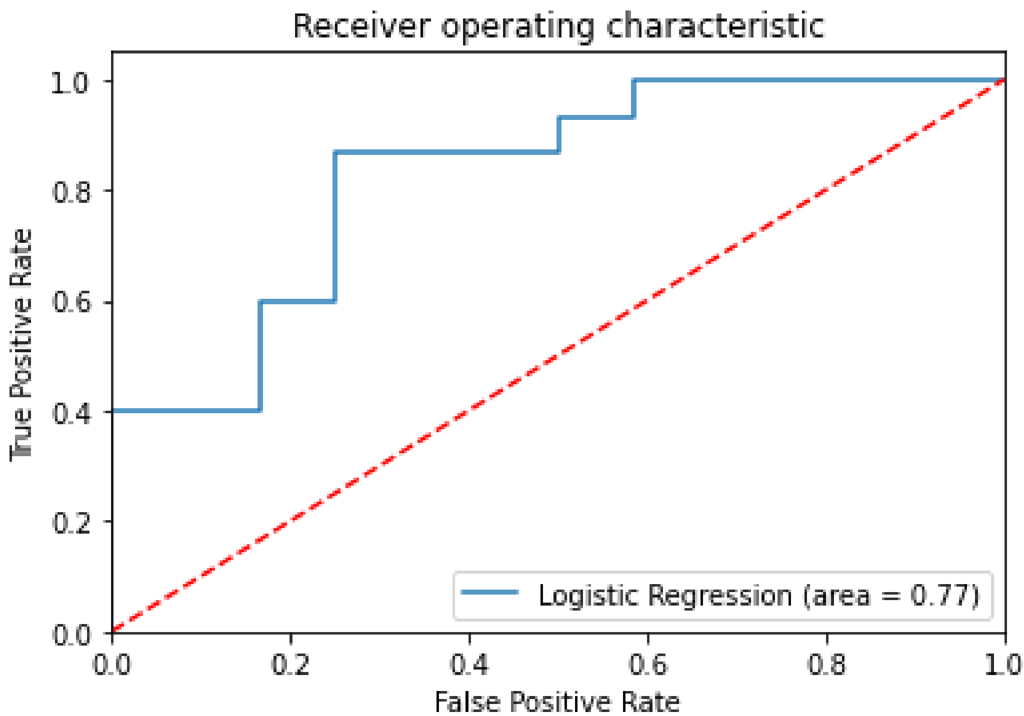
```
              precision    recall  f1-score   support

           0       0.80      0.67      0.73        12
           1       0.76      0.87      0.81        15

    accuracy                           0.78        27
   macro avg       0.78      0.77      0.77        27
weighted avg       0.78      0.78      0.77        27
```

[13]:
```
cnf_matrix = confusion_matrix(cnf_ytest, cnf_yhat, labels=[1,0])
plt.figure(figsize=(8,6))
sns.heatmap(cnf_matrix, square=True, annot=True, cbar=False)
plt.xlabel('Predicted Value')
plt.ylabel('True Value')
plt.show()
```



[14]:
```
logit_roc_auc = roc_auc_score(cnf_ytest, clf_lr.predict(cnf_xtest))
fpr, tpr, thresholds = roc_curve(cnf_ytest, clf_lr.predict_proba(cnf_xtest)[:
 ↪,1], pos_label=1)
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```



Advertising Stories

[15]:
```
temp_lst = []
for i in range(2, 6):
    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:
            clf_lr = LogisticRegression(penalty='l2', C=c, solver='liblinear')
            clf_lr.fit(X_train, y_train)
            y_hat = clf_lr.predict(X_test)
            y_hat_prob = clf_lr.predict_proba(X_test)
            temp_lst2 = []
            temp_lst2.append(i)
            temp_lst2.append(c)
            temp_lst2.append(metrics.accuracy_score(y_train, clf_lr.
 ↪predict(X_train)))
            temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
```

```
            temp_lst2.append(metrics.f1_score(y_test, y_hat))
            temp_lst2.append(metrics.jaccard_score(y_test, y_hat))
            temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst,
                       columns=['k', 'c', 'Train-set Accuracy', 'Test-set␣
 ↪Accuracy', 'F1 Score', 'Jaccard Score'])
temp_lst = []
for k in range(2, 6):
    for c in c_lst:
        temp_lst2 = []
        temp_lst2.append(k)
        temp_lst2.append(c)
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['c'] == c)]['Train-set Accuracy']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['c'] == c)]['Test-set Accuracy']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['c'] == c)]['F1 Score']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['c'] == c)]['Jaccard Score']), decimals=4))
        temp_lst.append(temp_lst2)

clf_lr_eval_df = pd.DataFrame(temp_lst,
                              columns=['k', 'c', 'Train-set Accuracy',␣
 ↪'Test-set Accuracy', 'F1 Score', 'Jaccard Score'])
clf_lr_eval_df
```

[15]:

	k	c	Train-set Accuracy	Test-set Accuracy	F1 Score	Jaccard Score
0	2	1.00	0.96	0.82	0.84	0.72
1	2	0.50	0.93	0.82	0.84	0.72
2	2	0.25	0.93	0.82	0.84	0.72
3	2	0.10	0.93	0.78	0.81	0.68
4	2	0.05	0.93	0.70	0.75	0.60
5	2	0.03	0.89	0.66	0.70	0.55
6	2	0.01	0.85	0.66	0.70	0.55
7	2	0.01	0.82	0.66	0.70	0.55
8	2	0.00	0.82	0.63	0.67	0.52
9	2	0.00	0.82	0.63	0.67	0.52
10	3	1.00	0.96	0.74	0.79	0.67
11	3	0.50	0.96	0.74	0.79	0.67
12	3	0.25	0.94	0.74	0.79	0.67
13	3	0.10	0.94	0.70	0.75	0.61
14	3	0.05	0.94	0.70	0.75	0.61
15	3	0.03	0.94	0.70	0.75	0.61
16	3	0.01	0.93	0.63	0.67	0.54
17	3	0.01	0.93	0.63	0.67	0.54

18	3	0.00	0.93	0.63	0.67	0.54
19	3	0.00	0.93	0.63	0.67	0.54
20	4	1.00	0.98	0.74	0.79	0.66
21	4	0.50	0.94	0.74	0.79	0.66
22	4	0.25	0.90	0.74	0.79	0.66
23	4	0.10	0.88	0.74	0.79	0.66
24	4	0.05	0.88	0.74	0.79	0.66
25	4	0.03	0.85	0.62	0.66	0.52
26	4	0.01	0.85	0.59	0.65	0.50
27	4	0.01	0.85	0.59	0.65	0.50
28	4	0.00	0.86	0.59	0.65	0.50
29	4	0.00	0.86	0.59	0.65	0.50
30	5	1.00	0.98	0.75	0.80	0.68
31	5	0.50	0.97	0.75	0.80	0.68
32	5	0.25	0.93	0.75	0.80	0.68
33	5	0.10	0.89	0.71	0.74	0.63
34	5	0.05	0.89	0.71	0.74	0.63
35	5	0.03	0.88	0.71	0.74	0.63
36	5	0.01	0.87	0.71	0.74	0.63
37	5	0.01	0.83	0.67	0.68	0.57
38	5	0.00	0.84	0.67	0.68	0.57
39	5	0.00	0.84	0.67	0.68	0.57

[16]:
```
clf_lr_eval_df.nlargest(3, 'Test-set Accuracy')
```

[16]:

	k	c	Train-set Accuracy	Test-set Accuracy	F1 Score	Jaccard Score
0	2	1.00	0.96	0.82	0.84	0.72
1	2	0.50	0.93	0.82	0.84	0.72
2	2	0.25	0.93	0.82	0.84	0.72

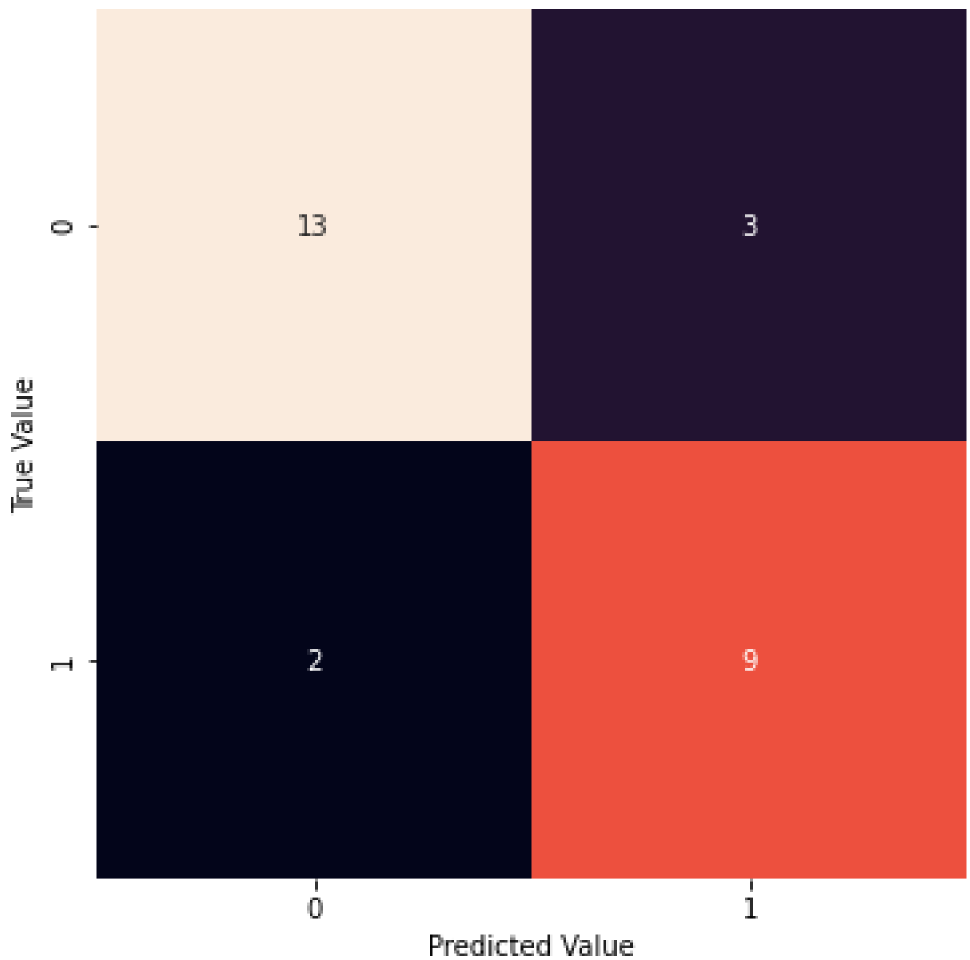
[17]:
```
kf = KFold(n_splits = 2)
temp_lst = []
clf_lr = LogisticRegression(penalty='l2', C=1, solver='liblinear')
for train_index, test_index in kf.split(ad_story_x):
    X_train, X_test = ad_story_x[train_index], ad_story_x[test_index]
    y_train, y_test = ad_story_y[train_index], ad_story_y[test_index]
    clf_lr.fit(X_train, y_train)
    y_hat = clf_lr.predict(X_test)
    y_hat_prob = clf_lr.predict_proba(X_test)
    temp_lst2 = []
    temp_lst2.append(metrics.accuracy_score(y_train, clf_lr.predict(X_train)))
    temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
    temp_lst2.append(metrics.f1_score(y_test, y_hat))
    temp_lst2.append(metrics.jaccard_score(y_test, y_hat))
    temp_lst2.append(metrics.log_loss(y_test, y_hat_prob))
    temp_lst2.append(y_test)
    temp_lst2.append(y_hat)
```

```
    temp_lst2.append(X_test)
    temp_lst.append(temp_lst2)
```

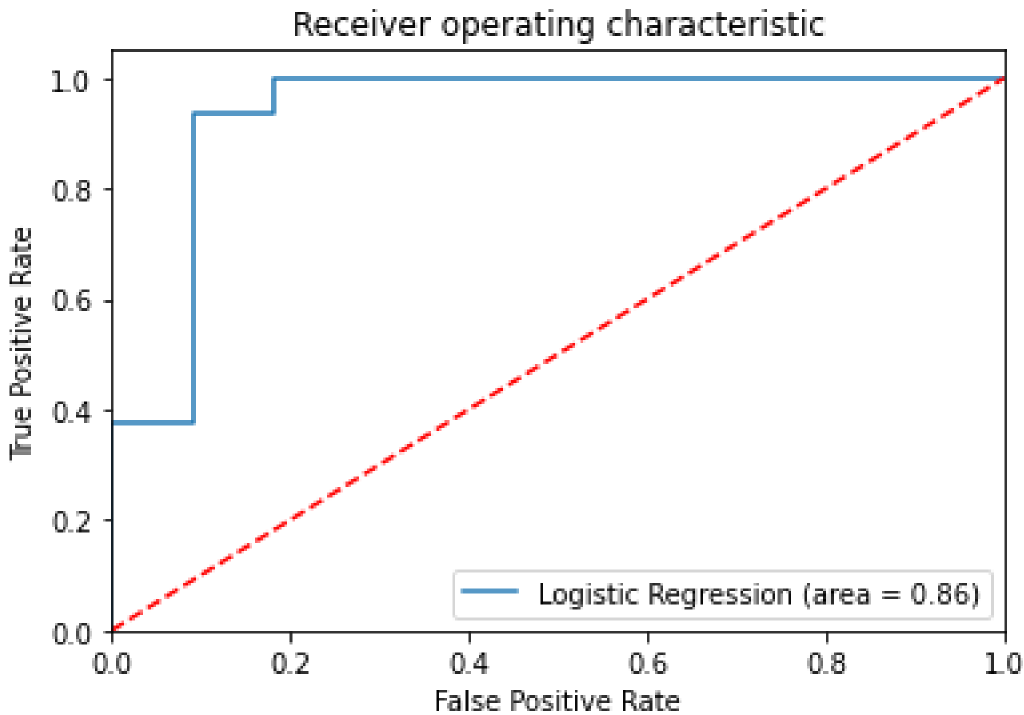
[18]:
```
temp_lst_ytest, temp_lst_yhat, temp_lst_xtest = [], [], []
for row in temp_lst:
    for i in row[5]:
        temp_lst_ytest.append(i)
    for j in row[6]:
        temp_lst_yhat.append(j)
    for k in row[7]:
        temp_lst_xtest.append(k)

cnf_ytest = np.array(temp_lst_ytest)
cnf_yhat = np.array(temp_lst_yhat)
cnf_xtest = np.array(temp_lst_xtest)
print(metrics.classification_report(cnf_ytest, cnf_yhat))
```

```
              precision    recall  f1-score   support

           0       0.75      0.82      0.78        11
           1       0.87      0.81      0.84        16

    accuracy                           0.81        27
   macro avg       0.81      0.82      0.81        27
weighted avg       0.82      0.81      0.82        27
```

[19]:
```
cnf_matrix = confusion_matrix(cnf_ytest, cnf_yhat, labels=[1,0])
plt.figure(figsize=(8,6))
sns.heatmap(cnf_matrix, square=True, annot=True, cbar=False)
plt.xlabel('Predicted Value')
plt.ylabel('True Value')
plt.show()
```



[20]:
```
logit_roc_auc = roc_auc_score(cnf_ytest, clf_lr.predict(cnf_xtest))
fpr, tpr, thresholds = roc_curve(cnf_ytest, clf_lr.predict_proba(cnf_xtest)[:
 ↪,1], pos_label=1)
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```



Influencer

[21]:
```
temp_lst = []
for i in range(2, 6):
    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:
            clf_lr = LogisticRegression(penalty='l2', C=c, solver='newton-cg')
            clf_lr.fit(X_train, y_train)
            y_hat = clf_lr.predict(X_test)
            y_hat_prob = clf_lr.predict_proba(X_test)
            temp_lst2 = []
            temp_lst2.append(i)
            temp_lst2.append(c)
            temp_lst2.append(metrics.accuracy_score(y_train, clf_lr.
 ↪predict(X_train)))
            temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
            temp_lst2.append(metrics.f1_score(y_test, y_hat, average='micro'))
            temp_lst2.append(metrics.jaccard_score(y_test, y_hat,␣
 ↪average='micro'))
            temp_lst.append(temp_lst2)
```

```python
temp_df = pd.DataFrame(temp_lst,
                       columns=['k', 'c', 'Train-set Accuracy', 'Test-set␣
 ↪Accuracy', 'F1 Score', 'Jaccard Score'])
temp_lst = []
for k in range(2, 6):
    for c in c_lst:
        temp_lst2 = []
        temp_lst2.append(k)
        temp_lst2.append(c)
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['c'] == c)]['Train-set Accuracy']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['c'] == c)]['Test-set Accuracy']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['c'] == c)]['F1 Score']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['c'] == c)]['Jaccard Score']), decimals=4))
        temp_lst.append(temp_lst2)

clf_lr_eval_df = pd.DataFrame(temp_lst,
                              columns=['k', 'c', 'Train-set Accuracy',␣
 ↪'Test-set Accuracy', 'F1 Score', 'Jaccard Score'])
clf_lr_eval_df
```

[21]:

	k	c	Train-set Accuracy	Test-set Accuracy	F1 Score	Jaccard Score
0	2	1.00	0.92	0.65	0.65	0.48
1	2	0.50	0.89	0.61	0.61	0.44
2	2	0.25	0.85	0.63	0.63	0.46
3	2	0.10	0.81	0.59	0.59	0.42
4	2	0.05	0.78	0.60	0.60	0.43
5	2	0.03	0.76	0.59	0.59	0.42
6	2	0.01	0.70	0.51	0.51	0.36
7	2	0.01	0.54	0.45	0.45	0.30
8	2	0.00	0.53	0.47	0.47	0.31
9	2	0.00	0.53	0.42	0.42	0.27
10	3	1.00	0.94	0.49	0.49	0.33
11	3	0.50	0.90	0.46	0.46	0.31
12	3	0.25	0.85	0.43	0.43	0.28
13	3	0.10	0.84	0.41	0.41	0.26
14	3	0.05	0.82	0.33	0.33	0.21
15	3	0.03	0.77	0.32	0.32	0.20
16	3	0.01	0.64	0.27	0.27	0.16
17	3	0.01	0.60	0.23	0.23	0.13
18	3	0.00	0.51	0.21	0.21	0.12
19	3	0.00	0.50	0.21	0.21	0.12
20	4	1.00	0.92	0.56	0.56	0.40
21	4	0.50	0.88	0.55	0.55	0.39

```
22  4 0.25                0.85               0.48      0.48           0.33
23  4 0.10                0.79               0.47      0.47           0.32
24  4 0.05                0.77               0.42      0.42           0.28
25  4 0.03                0.74               0.42      0.42           0.27
26  4 0.01                0.68               0.41      0.41           0.28
27  4 0.01                0.62               0.23      0.23           0.13
28  4 0.00                0.57               0.20      0.20           0.11
29  4 0.00                0.53               0.20      0.20           0.11
30  5 1.00                0.93               0.63      0.63           0.51
31  5 0.50                0.90               0.57      0.57           0.44
32  5 0.25                0.85               0.45      0.45           0.32
33  5 0.10                0.80               0.47      0.47           0.36
34  5 0.05                0.76               0.48      0.48           0.37
35  5 0.03                0.73               0.46      0.46           0.35
36  5 0.01                0.71               0.51      0.51           0.38
37  5 0.01                0.60               0.41      0.41           0.28
38  5 0.00                0.57               0.37      0.37           0.25
39  5 0.00                0.51               0.30      0.30           0.18
```

[22]:
```python
clf_lr_eval_df.nlargest(3, 'Test-set Accuracy')
```

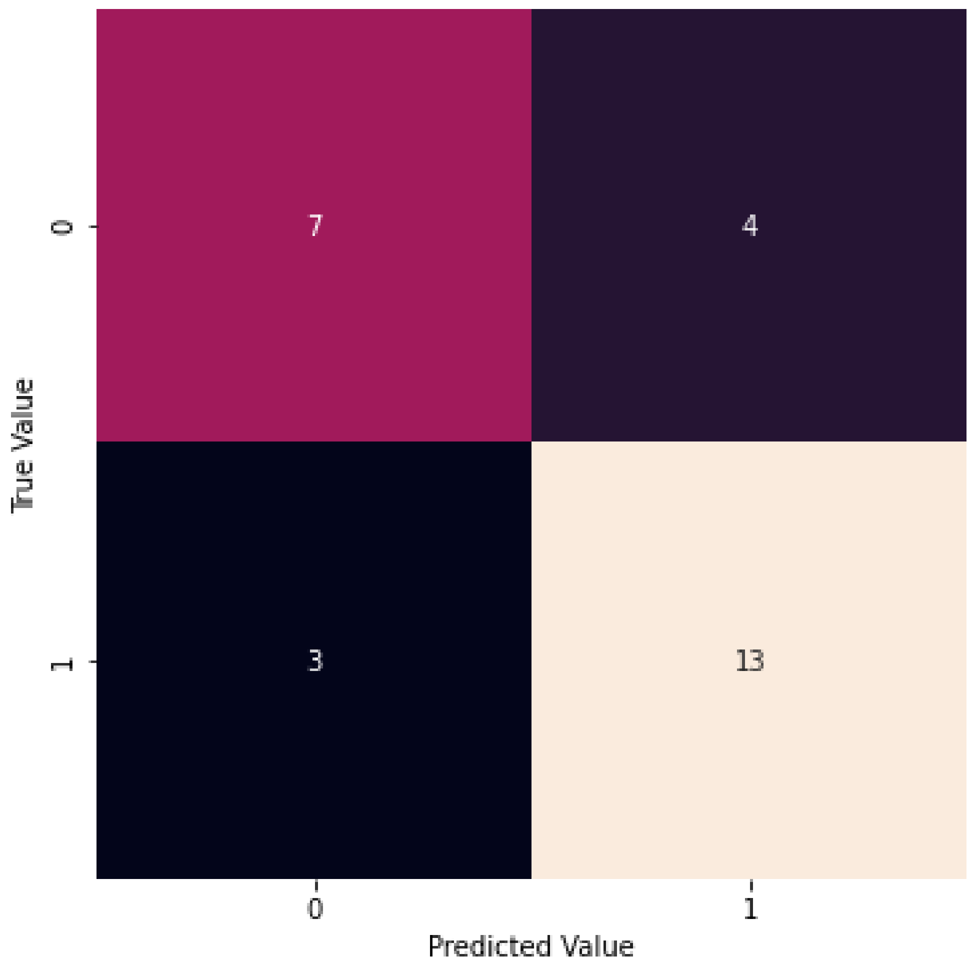
[22]:
```
    k    c  Train-set Accuracy  Test-set Accuracy  F1 Score  Jaccard Score
0   2 1.00                0.92               0.65      0.65           0.48
30  5 1.00                0.93               0.63      0.63           0.51
2   2 0.25                0.85               0.63      0.63           0.46
```

[23]:
```python
kf = KFold(n_splits = 2)
temp_lst = []
clf_lr = LogisticRegression(penalty='l2', C=1, solver='newton-cg')
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]
    clf_lr.fit(X_train, y_train)
    y_hat = clf_lr.predict(X_test)
    y_hat_prob = clf_lr.predict_proba(X_test)
    temp_lst2 = []
    temp_lst2.append(metrics.accuracy_score(y_train, clf_lr.predict(X_train)))
    temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
    temp_lst2.append(metrics.f1_score(y_test, y_hat, average='micro'))
    temp_lst2.append(metrics.jaccard_score(y_test, y_hat, average='micro'))
    temp_lst2.append(metrics.log_loss(y_test, y_hat_prob))
    temp_lst2.append(y_test)
    temp_lst2.append(y_hat)
    temp_lst2.append(X_test)
    temp_lst.append(temp_lst2)
```

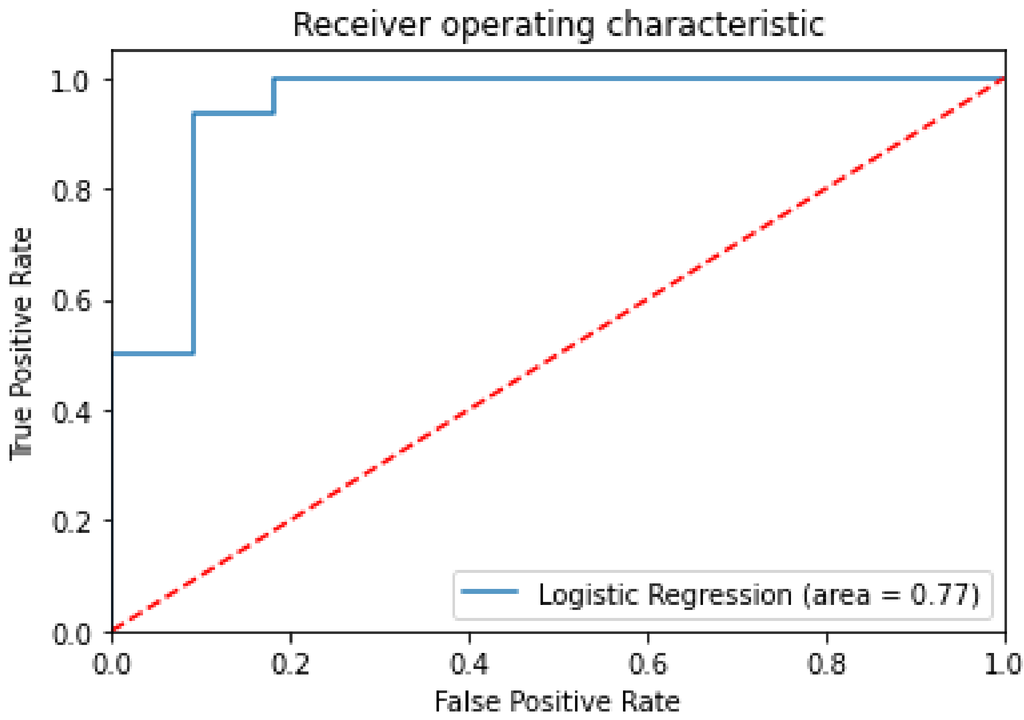
[24]:
```
temp_lst_ytest, temp_lst_yhat, temp_lst_xtest = [], [], []
for row in temp_lst:
    for i in row[5]:
        temp_lst_ytest.append(i)
    for j in row[6]:
        temp_lst_yhat.append(j)
    for k in row[7]:
        temp_lst_xtest.append(k)

cnf_ytest = np.array(temp_lst_ytest)
cnf_yhat = np.array(temp_lst_yhat)
cnf_xtest = np.array(temp_lst_xtest)
print(metrics.classification_report(cnf_ytest, cnf_yhat))
```

```
              precision    recall  f1-score   support

           0       0.70      0.64      0.67        11
           1       0.76      0.81      0.79        16

    accuracy                           0.74        27
   macro avg       0.73      0.72      0.73        27
weighted avg       0.74      0.74      0.74        27
```

[25]:
```
cnf_matrix = confusion_matrix(cnf_ytest, cnf_yhat)
plt.figure(figsize=(8,6))
sns.heatmap(cnf_matrix, square=True, annot=True, cbar=False)
plt.xlabel('Predicted Value')
plt.ylabel('True Value')
plt.show()
```



[26]:
```
logit_roc_auc = roc_auc_score(cnf_ytest, clf_lr.predict(cnf_xtest))
fpr, tpr, thresholds = roc_curve(cnf_ytest, clf_lr.predict_proba(cnf_xtest)[:
 ↪,1], pos_label=1)
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```



Leaders Post

[27]:
```
temp_lst = []
for i in range(2, 6):
    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:
            clf_lr = LogisticRegression(penalty='l2', C=c, solver='newton-cg')
            clf_lr.fit(X_train, y_train)
            y_hat = clf_lr.predict(X_test)
            y_hat_prob = clf_lr.predict_proba(X_test)
            temp_lst2 = []
            temp_lst2.append(i)
            temp_lst2.append(c)
            temp_lst2.append(metrics.accuracy_score(y_train, clf_lr.
 ↪predict(X_train)))
            temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
            temp_lst2.append(metrics.f1_score(y_test, y_hat, average='micro'))
            temp_lst2.append(metrics.jaccard_score(y_test, y_hat,␣
 ↪average='micro'))
```

```
            temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst,
                       columns=['k', 'c', 'Train-set Accuracy', 'Test-set Accuracy', 'F1 Score', 'Jaccard Score'])
temp_lst = []
for k in range(2, 6):
    for c in c_lst:
        temp_lst2 = []
        temp_lst2.append(k)
        temp_lst2.append(c)
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) & (temp_df['c'] == c)]['Train-set Accuracy']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) & (temp_df['c'] == c)]['Test-set Accuracy']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) & (temp_df['c'] == c)]['F1 Score']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) & (temp_df['c'] == c)]['Jaccard Score']), decimals=4))
        temp_lst.append(temp_lst2)

clf_lr_eval_df = pd.DataFrame(temp_lst,
                              columns=['k', 'c', 'Train-set Accuracy', 'Test-set Accuracy', 'F1 Score', 'Jaccard Score'])
clf_lr_eval_df
```

[27]:

	k	c	Train-set Accuracy	Test-set Accuracy	F1 Score	Jaccard Score
0	2	1.00	1.00	0.35	0.35	0.22
1	2	0.50	1.00	0.35	0.35	0.22
2	2	0.25	0.88	0.45	0.45	0.29
3	2	0.10	0.88	0.45	0.45	0.29
4	2	0.05	0.78	0.45	0.45	0.29
5	2	0.03	0.78	0.45	0.45	0.29
6	2	0.01	0.78	0.45	0.45	0.29
7	2	0.01	0.78	0.45	0.45	0.29
8	2	0.00	0.78	0.45	0.45	0.29
9	2	0.00	0.78	0.45	0.45	0.29
10	3	1.00	1.00	0.44	0.44	0.30
11	3	0.50	0.89	0.44	0.44	0.30
12	3	0.25	0.89	0.33	0.33	0.20
13	3	0.10	0.83	0.33	0.33	0.20
14	3	0.05	0.78	0.33	0.33	0.20
15	3	0.03	0.72	0.33	0.33	0.20
16	3	0.01	0.72	0.44	0.44	0.30
17	3	0.01	0.72	0.44	0.44	0.30
18	3	0.00	0.72	0.44	0.44	0.30
19	3	0.00	0.72	0.44	0.44	0.30

20	4	1.00	1.00	0.42	0.42	0.29
21	4	0.50	0.96	0.42	0.42	0.29
22	4	0.25	0.85	0.46	0.46	0.30
23	4	0.10	0.82	0.46	0.46	0.30
24	4	0.05	0.77	0.46	0.46	0.30
25	4	0.03	0.70	0.46	0.46	0.30
26	4	0.01	0.60	0.58	0.58	0.47
27	4	0.01	0.56	0.58	0.58	0.47
28	4	0.00	0.56	0.58	0.58	0.47
29	4	0.00	0.56	0.58	0.58	0.47
30	5	1.00	1.00	0.50	0.50	0.40
31	5	0.50	0.89	0.50	0.50	0.40
32	5	0.25	0.84	0.50	0.50	0.40
33	5	0.10	0.81	0.50	0.50	0.40
34	5	0.05	0.81	0.50	0.50	0.40
35	5	0.03	0.72	0.50	0.50	0.40
36	5	0.01	0.59	0.60	0.60	0.53
37	5	0.01	0.56	0.60	0.60	0.53
38	5	0.00	0.56	0.60	0.60	0.53
39	5	0.00	0.56	0.60	0.60	0.53

[28]:

```
clf_lr_eval_df.nlargest(3, 'Test-set Accuracy')
```

[28]:

	k	c	Train-set Accuracy	Test-set Accuracy	F1 Score	Jaccard Score
36	5	0.01	0.59	0.60	0.60	0.53
37	5	0.01	0.56	0.60	0.60	0.53
38	5	0.00	0.56	0.60	0.60	0.53

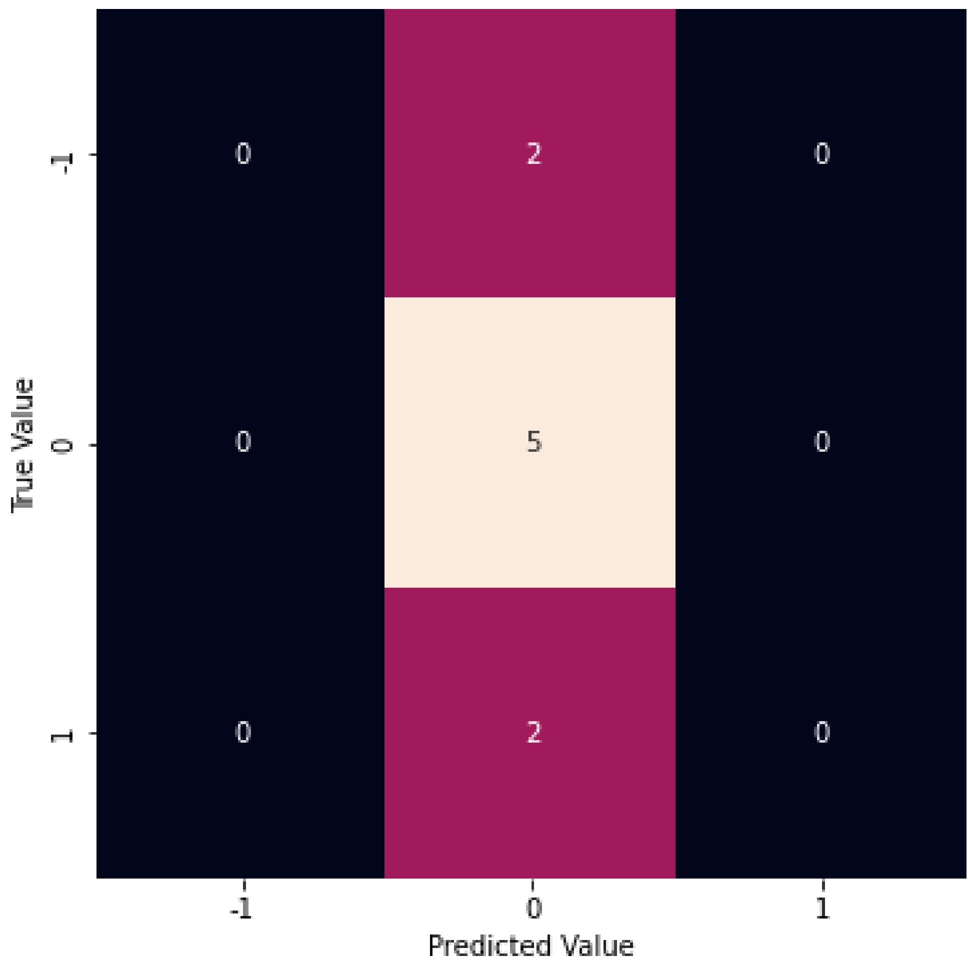
[29]:

```
kf = KFold(n_splits = 5)
temp_lst = []
clf_lr = LogisticRegression(penalty='l2', C=.01 ,solver='newton-cg')
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]
    clf_lr.fit(X_train, y_train)
    y_hat = clf_lr.predict(X_test)
    y_hat_prob = clf_lr.predict_proba(X_test)
    temp_lst2 = []
    temp_lst2.append(metrics.accuracy_score(y_train, clf_lr.predict(X_train)))
    temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
    temp_lst2.append(metrics.f1_score(y_test, y_hat, average='micro'))
    temp_lst2.append(metrics.jaccard_score(y_test, y_hat, average='micro'))
    temp_lst2.append(0)
    temp_lst2.append(y_test)
    temp_lst2.append(y_hat)
    temp_lst2.append(X_test)
    temp_lst.append(temp_lst2)
```

[30]:
```
temp_lst_ytest, temp_lst_yhat, temp_lst_xtest = [], [], []
for row in temp_lst:
    for i in row[5]:
        temp_lst_ytest.append(i)
    for j in row[6]:
        temp_lst_yhat.append(j)
    for k in row[7]:
        temp_lst_xtest.append(k)

cnf_ytest = np.array(temp_lst_ytest)
cnf_yhat = np.array(temp_lst_yhat)
cnf_xtest = np.array(temp_lst_xtest)
print(metrics.classification_report(cnf_ytest, cnf_yhat))
```

```
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         2
           0       0.56      1.00      0.71         5
           1       0.00      0.00      0.00         2

    accuracy                           0.56         9
   macro avg       0.19      0.33      0.24         9
weighted avg       0.31      0.56      0.40         9
```

[31]:
```
cnf_matrix = confusion_matrix(cnf_ytest, cnf_yhat)
plt.figure(figsize=(8,6))
sns.heatmap(cnf_matrix, square=True, annot=True, cbar=False)
plt.xticks([.5, 1.5, 2.5], [-1, 0, 1])
plt.yticks([.5, 1.5, 2.5], [-1, 0, 1])
plt.xlabel('Predicted Value')
plt.ylabel('True Value')
plt.show()
```



1.1.2 Support Vector Machine (Both)

Advertising Post

[32]:
```
from sklearn.svm import SVC
```

[33]:
```
c_lst = [1, .5, .25, .1, .05, .025, .01, .005, .0025, .001]
kernel_lst = ['linear', 'poly', 'rbf', 'sigmoid']
```

[34]:
```
temp_lst = []
for i in range(2, 6):
    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:
                clf_svm = SVC(C=c, kernel=kernel_type)
```

```
                clf_svm.fit(X_train, y_train)
                y_hat = clf_svm.predict(X_test)
                temp_lst2 = []
                temp_lst2.append(i)
                temp_lst2.append(c)
                temp_lst2.append(kernel_type)
                temp_lst2.append(metrics.accuracy_score(y_train, clf_svm.
 ↪predict(X_train)))
                temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
                temp_lst2.append(metrics.f1_score(y_test, y_hat))
                temp_lst2.append(metrics.jaccard_score(y_test, y_hat))
                temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst,
                       columns=['k', 'c', 'kernel', 'Train-set Accuracy',␣
 ↪'Test-set Accuracy', 'F1 Score', 'Jaccard Score'])
temp_lst = []
for k in range(2, 6):
    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-set␣
 ↪Accuracy']), 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-set␣
 ↪Accuracy']), decimals=4))
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['c'] == c) & (temp_df['kernel'] == kernel_type)]['F1 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)]['Jaccard Score']),␣
 ↪decimals=4))
            temp_lst.append(temp_lst2)

clf_svm_eval_df = pd.DataFrame(temp_lst,
                              columns=['k', 'c', 'kernel', 'Train-set␣
 ↪Accuracy', 'Test-set Accuracy', 'F1 Score', 'Jaccard Score'])
clf_svm_eval_df
```

[34]:

```
     k    c   kernel  Train-set Accuracy  Test-set Accuracy  F1 Score  \
0    2 1.00   linear                0.81               0.70      0.76
1    2 1.00     poly                0.77               0.45      0.33
```

2	2	1.00	rbf	0.81	0.63	0.69
3	2	1.00	sigmoid	0.77	0.52	0.63
4	2	0.50	linear	0.81	0.59	0.70
5	2	0.50	poly	0.77	0.45	0.33
6	2	0.50	rbf	0.73	0.45	0.33
7	2	0.50	sigmoid	0.77	0.55	0.65
8	2	0.25	linear	0.85	0.55	0.67
9	2	0.25	poly	0.77	0.45	0.33
10	2	0.25	rbf	0.70	0.41	0.32
11	2	0.25	sigmoid	0.70	0.41	0.32
12	2	0.10	linear	0.81	0.63	0.69
13	2	0.10	poly	0.73	0.45	0.33
14	2	0.10	rbf	0.59	0.41	0.32
15	2	0.10	sigmoid	0.59	0.41	0.32
16	2	0.05	linear	0.74	0.41	0.32
17	2	0.05	poly	0.63	0.45	0.33
18	2	0.05	rbf	0.59	0.41	0.32
19	2	0.05	sigmoid	0.59	0.41	0.32
20	2	0.03	linear	0.63	0.41	0.32
21	2	0.03	poly	0.63	0.41	0.32
22	2	0.03	rbf	0.59	0.41	0.32
23	2	0.03	sigmoid	0.59	0.41	0.32
24	2	0.01	linear	0.59	0.41	0.32
25	2	0.01	poly	0.59	0.41	0.32
26	2	0.01	rbf	0.59	0.41	0.32
27	2	0.01	sigmoid	0.59	0.41	0.32
28	2	0.01	linear	0.59	0.41	0.32
29	2	0.01	poly	0.59	0.41	0.32
30	2	0.01	rbf	0.59	0.41	0.32
31	2	0.01	sigmoid	0.59	0.41	0.32
32	2	0.00	linear	0.59	0.41	0.32
33	2	0.00	poly	0.59	0.41	0.32
34	2	0.00	rbf	0.59	0.41	0.32
35	2	0.00	sigmoid	0.59	0.41	0.32
36	2	0.00	linear	0.59	0.41	0.32
37	2	0.00	poly	0.59	0.41	0.32
38	2	0.00	rbf	0.59	0.41	0.32
39	2	0.00	sigmoid	0.59	0.41	0.32
40	3	1.00	linear	0.89	0.74	0.77
41	3	1.00	poly	0.78	0.70	0.78
42	3	1.00	rbf	0.78	0.63	0.70
43	3	1.00	sigmoid	0.76	0.78	0.72
44	3	0.50	linear	0.85	0.74	0.77
45	3	0.50	poly	0.80	0.52	0.55
46	3	0.50	rbf	0.69	0.33	0.36
47	3	0.50	sigmoid	0.72	0.48	0.43
48	3	0.25	linear	0.80	0.70	0.74

49	3	0.25	poly	0.72	0.41	0.42
50	3	0.25	rbf	0.69	0.37	0.40
51	3	0.25	sigmoid	0.65	0.41	0.42
52	3	0.10	linear	0.78	0.70	0.76
53	3	0.10	poly	0.67	0.41	0.42
54	3	0.10	rbf	0.59	0.37	0.40
55	3	0.10	sigmoid	0.59	0.37	0.40
56	3	0.05	linear	0.78	0.44	0.41
57	3	0.05	poly	0.63	0.41	0.42
58	3	0.05	rbf	0.59	0.37	0.40
59	3	0.05	sigmoid	0.59	0.37	0.40
60	3	0.03	linear	0.69	0.41	0.42
61	3	0.03	poly	0.63	0.37	0.40
62	3	0.03	rbf	0.59	0.37	0.40
63	3	0.03	sigmoid	0.59	0.37	0.40
64	3	0.01	linear	0.59	0.37	0.40
65	3	0.01	poly	0.59	0.37	0.40
66	3	0.01	rbf	0.59	0.37	0.40
67	3	0.01	sigmoid	0.59	0.37	0.40
68	3	0.01	linear	0.59	0.37	0.40
69	3	0.01	poly	0.59	0.37	0.40
70	3	0.01	rbf	0.59	0.37	0.40
71	3	0.01	sigmoid	0.59	0.37	0.40
72	3	0.00	linear	0.59	0.37	0.40
73	3	0.00	poly	0.59	0.37	0.40
74	3	0.00	rbf	0.59	0.37	0.40
75	3	0.00	sigmoid	0.59	0.37	0.40
76	3	0.00	linear	0.59	0.37	0.40
77	3	0.00	poly	0.59	0.37	0.40
78	3	0.00	rbf	0.59	0.37	0.40
79	3	0.00	sigmoid	0.59	0.37	0.40
80	4	1.00	linear	0.85	0.69	0.71
81	4	1.00	poly	0.79	0.68	0.74
82	4	1.00	rbf	0.80	0.58	0.65
83	4	1.00	sigmoid	0.76	0.69	0.71
84	4	0.50	linear	0.86	0.65	0.70
85	4	0.50	poly	0.79	0.68	0.74
86	4	0.50	rbf	0.75	0.47	0.59
87	4	0.50	sigmoid	0.78	0.54	0.63
88	4	0.25	linear	0.84	0.65	0.70
89	4	0.25	poly	0.75	0.54	0.63
90	4	0.25	rbf	0.70	0.33	0.39
91	4	0.25	sigmoid	0.67	0.36	0.40
92	4	0.10	linear	0.80	0.65	0.70
93	4	0.10	poly	0.67	0.43	0.47
94	4	0.10	rbf	0.67	0.33	0.39
95	4	0.10	sigmoid	0.67	0.33	0.39

96	4	0.05	linear	0.75	0.61	0.69
97	4	0.05	poly	0.63	0.43	0.47
98	4	0.05	rbf	0.67	0.33	0.39
99	4	0.05	sigmoid	0.67	0.33	0.39
100	4	0.03	linear	0.64	0.40	0.44
101	4	0.03	poly	0.60	0.40	0.45
102	4	0.03	rbf	0.67	0.33	0.39
103	4	0.03	sigmoid	0.67	0.33	0.39
104	4	0.01	linear	0.64	0.36	0.42
105	4	0.01	poly	0.59	0.40	0.45
106	4	0.01	rbf	0.67	0.33	0.39
107	4	0.01	sigmoid	0.67	0.33	0.39
108	4	0.01	linear	0.64	0.36	0.42
109	4	0.01	poly	0.59	0.40	0.45
110	4	0.01	rbf	0.67	0.33	0.39
111	4	0.01	sigmoid	0.67	0.33	0.39
112	4	0.00	linear	0.64	0.36	0.42
113	4	0.00	poly	0.59	0.40	0.45
114	4	0.00	rbf	0.67	0.33	0.39
115	4	0.00	sigmoid	0.67	0.33	0.39
116	4	0.00	linear	0.64	0.36	0.42
117	4	0.00	poly	0.59	0.40	0.45
118	4	0.00	rbf	0.67	0.33	0.39
119	4	0.00	sigmoid	0.67	0.33	0.39
120	5	1.00	linear	0.84	0.73	0.72
121	5	1.00	poly	0.79	0.65	0.72
122	5	1.00	rbf	0.80	0.55	0.61
123	5	1.00	sigmoid	0.79	0.55	0.61
124	5	0.50	linear	0.81	0.58	0.64
125	5	0.50	poly	0.79	0.65	0.72
126	5	0.50	rbf	0.77	0.45	0.55
127	5	0.50	sigmoid	0.76	0.48	0.56
128	5	0.25	linear	0.80	0.55	0.61
129	5	0.25	poly	0.74	0.66	0.73
130	5	0.25	rbf	0.74	0.45	0.55
131	5	0.25	sigmoid	0.72	0.49	0.56
132	5	0.10	linear	0.78	0.59	0.66
133	5	0.10	poly	0.66	0.63	0.71
134	5	0.10	rbf	0.61	0.46	0.52
135	5	0.10	sigmoid	0.60	0.46	0.52
136	5	0.05	linear	0.74	0.51	0.60
137	5	0.05	poly	0.63	0.49	0.53
138	5	0.05	rbf	0.61	0.46	0.52
139	5	0.05	sigmoid	0.60	0.46	0.52
140	5	0.03	linear	0.69	0.52	0.63
141	5	0.03	poly	0.61	0.46	0.52
142	5	0.03	rbf	0.61	0.46	0.52

143	5	0.03	sigmoid	0.60	0.46	0.52
144	5	0.01	linear	0.60	0.46	0.52
145	5	0.01	poly	0.57	0.46	0.52
146	5	0.01	rbf	0.61	0.46	0.52
147	5	0.01	sigmoid	0.60	0.46	0.52
148	5	0.01	linear	0.59	0.46	0.52
149	5	0.01	poly	0.57	0.46	0.52
150	5	0.01	rbf	0.61	0.46	0.52
151	5	0.01	sigmoid	0.60	0.46	0.52
152	5	0.00	linear	0.59	0.46	0.52
153	5	0.00	poly	0.57	0.46	0.52
154	5	0.00	rbf	0.61	0.46	0.52
155	5	0.00	sigmoid	0.60	0.46	0.52
156	5	0.00	linear	0.59	0.46	0.52
157	5	0.00	poly	0.57	0.46	0.52
158	5	0.00	rbf	0.61	0.46	0.52
159	5	0.00	sigmoid	0.60	0.46	0.52

	Jaccard Score
0	0.62
1	0.25
2	0.53
3	0.46
4	0.54
5	0.25
6	0.25
7	0.48
8	0.50
9	0.25
10	0.23
11	0.23
12	0.53
13	0.25
14	0.23
15	0.23
16	0.23
17	0.25
18	0.23
19	0.23
20	0.23
21	0.23
22	0.23
23	0.23
24	0.23
25	0.23
26	0.23
27	0.23

28	0.23
29	0.23
30	0.23
31	0.23
32	0.23
33	0.23
34	0.23
35	0.23
36	0.23
37	0.23
38	0.23
39	0.23
40	0.69
41	0.68
42	0.59
43	0.63
44	0.69
45	0.44
46	0.26
47	0.33
48	0.65
49	0.32
50	0.30
51	0.32
52	0.69
53	0.32
54	0.30
55	0.30
56	0.32
57	0.32
58	0.30
59	0.30
60	0.32
61	0.30
62	0.30
63	0.30
64	0.30
65	0.30
66	0.30
67	0.30
68	0.30
69	0.30
70	0.30
71	0.30
72	0.30
73	0.30
74	0.30

75	0.30
76	0.30
77	0.30
78	0.30
79	0.30
80	0.61
81	0.65
82	0.52
83	0.61
84	0.60
85	0.65
86	0.44
87	0.48
88	0.60
89	0.50
90	0.27
91	0.29
92	0.60
93	0.38
94	0.27
95	0.27
96	0.57
97	0.38
98	0.27
99	0.27
100	0.33
101	0.36
102	0.27
103	0.27
104	0.32
105	0.36
106	0.27
107	0.27
108	0.32
109	0.36
110	0.27
111	0.27
112	0.32
113	0.36
114	0.27
115	0.27
116	0.32
117	0.36
118	0.27
119	0.27
120	0.63
121	0.61

```
122           0.49
123           0.49
124           0.53
125           0.61
126           0.43
127           0.44
128           0.49
129           0.64
130           0.43
131           0.44
132           0.53
133           0.61
134           0.43
135           0.43
136           0.48
137           0.44
138           0.43
139           0.43
140           0.51
141           0.43
142           0.43
143           0.43
144           0.43
145           0.43
146           0.43
147           0.43
148           0.43
149           0.43
150           0.43
151           0.43
152           0.43
153           0.43
154           0.43
155           0.43
156           0.43
157           0.43
158           0.43
159           0.43
```

[35]:
```
clf_svm_eval_df.nlargest(3, 'Test-set Accuracy')
```

[35]:
```
    k    c   kernel  Train-set Accuracy  Test-set Accuracy  F1 Score  \
43  3 1.00  sigmoid                0.76               0.78      0.72
40  3 1.00   linear                0.89               0.74      0.77
44  3 0.50   linear                0.85               0.74      0.77

    Jaccard Score
```

```
43          0.63
40          0.69
44          0.69
```

[36]:

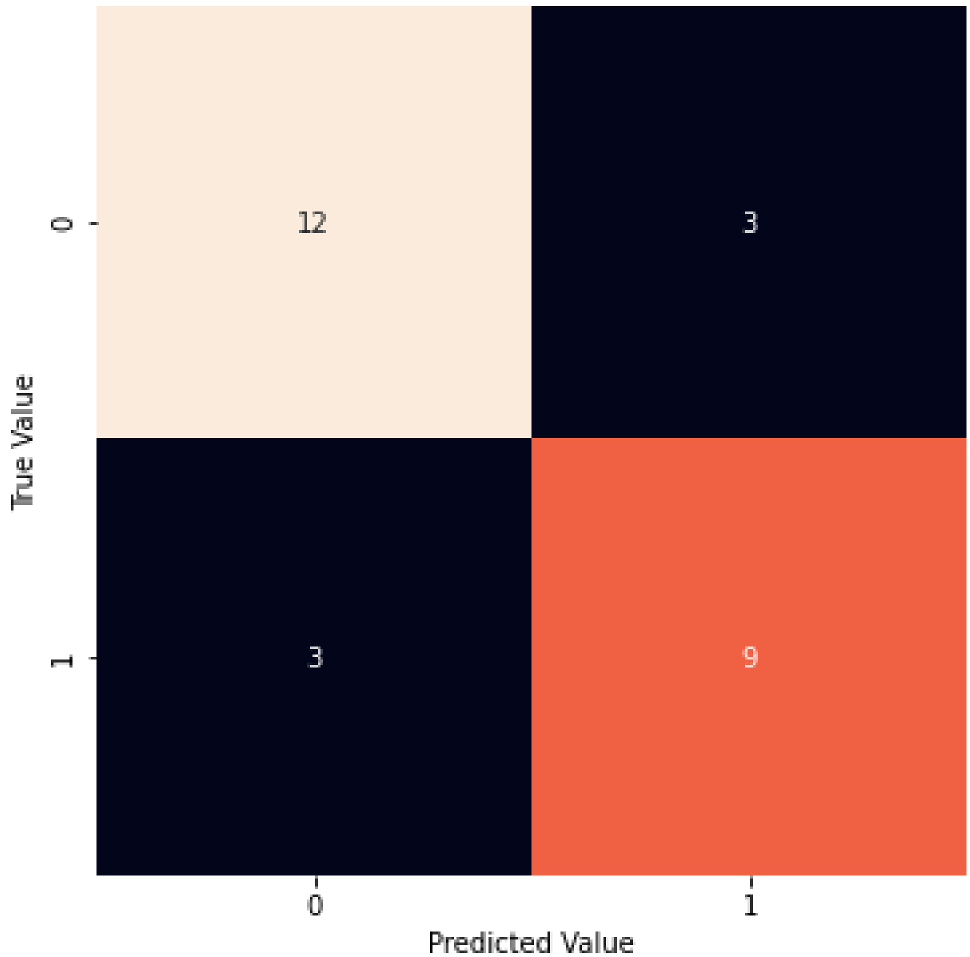
```python
kf = KFold(n_splits = 3)
temp_lst = []
clf_svm = SVC(C=1, kernel='sigmoid')
for train_index, test_index in kf.split(ad_post_x):
    X_train, X_test = ad_post_x[train_index], ad_post_x[test_index]
    y_train, y_test = ad_post_y[train_index], ad_post_y[test_index]
    clf_svm.fit(X_train, y_train)
    y_hat = clf_svm.predict(X_test)
    temp_lst2 = []
    temp_lst2.append(metrics.accuracy_score(y_train, clf_svm.predict(X_train)))
    temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
    temp_lst2.append(metrics.f1_score(y_test, y_hat))
    temp_lst2.append(metrics.jaccard_score(y_test, y_hat))
    temp_lst2.append(y_test)
    temp_lst2.append(y_hat)
    temp_lst2.append(X_test)
    temp_lst.append(temp_lst2)
```

[37]:

```python
temp_lst_ytest, temp_lst_yhat, temp_lst_xtest = [], [], []
for row in temp_lst:
    for i in row[4]:
        temp_lst_ytest.append(i)
    for j in row[5]:
        temp_lst_yhat.append(j)
    for k in row[6]:
        temp_lst_xtest.append(k)

cnf_ytest = np.array(temp_lst_ytest)
cnf_yhat = np.array(temp_lst_yhat)
cnf_xtest = np.array(temp_lst_xtest)
print(metrics.classification_report(cnf_ytest, cnf_yhat))
```

```
              precision    recall  f1-score   support

           0       0.75      0.75      0.75        12
           1       0.80      0.80      0.80        15

    accuracy                           0.78        27
   macro avg       0.78      0.78      0.78        27
weighted avg       0.78      0.78      0.78        27
```

[38]:
```
cnf_matrix = confusion_matrix(cnf_ytest, cnf_yhat, labels=[1,0])
plt.figure(figsize=(8,6))
sns.heatmap(cnf_matrix, square=True, annot=True, cbar=False)
plt.xlabel('Predicted Value')
plt.ylabel('True Value')
plt.show()
```



Advertising Stories

[39]:
```
temp_lst = []
for i in range(2, 6):
    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:
```

```
                clf_svm = SVC(C=c, kernel=kernel_type)
                clf_svm.fit(X_train, y_train)
                y_hat = clf_svm.predict(X_test)
                temp_lst2 = []
                temp_lst2.append(i)
                temp_lst2.append(c)
                temp_lst2.append(kernel_type)
                temp_lst2.append(metrics.accuracy_score(y_train, clf_svm.
 ↪predict(X_train)))
                temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
                temp_lst2.append(metrics.f1_score(y_test, y_hat))
                temp_lst2.append(metrics.jaccard_score(y_test, y_hat))
                temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst,
                       columns=['k', 'c', 'kernel', 'Train-set Accuracy',␣
 ↪'Test-set Accuracy', 'F1 Score', 'Jaccard Score'])
temp_lst = []
for k in range(2, 6):
    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-set␣
 ↪Accuracy']), 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-set␣
 ↪Accuracy']), decimals=4))
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['c'] == c) & (temp_df['kernel'] == kernel_type)]['F1 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)]['Jaccard Score']),␣
 ↪decimals=4))
            temp_lst.append(temp_lst2)

clf_svm_eval_df = pd.DataFrame(temp_lst,
                               columns=['k', 'c', 'kernel', 'Train-set␣
 ↪Accuracy', 'Test-set Accuracy', 'F1 Score', 'Jaccard Score'])
clf_svm_eval_df
```

[39]:

```
    k    c   kernel  Train-set Accuracy  Test-set Accuracy  F1 Score  \
0   2 1.00   linear                0.96               0.82      0.84
```

1	2	1.00	poly	0.89	0.66	0.75
2	2	1.00	rbf	0.93	0.70	0.76
3	2	1.00	sigmoid	0.85	0.63	0.72
4	2	0.50	linear	0.96	0.82	0.84
5	2	0.50	poly	0.81	0.52	0.56
6	2	0.50	rbf	0.82	0.55	0.65
7	2	0.50	sigmoid	0.74	0.59	0.70
8	2	0.25	linear	0.93	0.82	0.84
9	2	0.25	poly	0.70	0.41	0.33
10	2	0.25	rbf	0.63	0.37	0.32
11	2	0.25	sigmoid	0.63	0.37	0.32
12	2	0.10	linear	0.93	0.70	0.76
13	2	0.10	poly	0.70	0.41	0.33
14	2	0.10	rbf	0.63	0.37	0.32
15	2	0.10	sigmoid	0.63	0.37	0.32
16	2	0.05	linear	0.85	0.59	0.70
17	2	0.05	poly	0.66	0.41	0.33
18	2	0.05	rbf	0.63	0.37	0.32
19	2	0.05	sigmoid	0.63	0.37	0.32
20	2	0.03	linear	0.63	0.37	0.32
21	2	0.03	poly	0.66	0.41	0.33
22	2	0.03	rbf	0.63	0.37	0.32
23	2	0.03	sigmoid	0.63	0.37	0.32
24	2	0.01	linear	0.63	0.37	0.32
25	2	0.01	poly	0.63	0.37	0.32
26	2	0.01	rbf	0.63	0.37	0.32
27	2	0.01	sigmoid	0.63	0.37	0.32
28	2	0.01	linear	0.63	0.37	0.32
29	2	0.01	poly	0.63	0.37	0.32
30	2	0.01	rbf	0.63	0.37	0.32
31	2	0.01	sigmoid	0.63	0.37	0.32
32	2	0.00	linear	0.63	0.37	0.32
33	2	0.00	poly	0.63	0.37	0.32
34	2	0.00	rbf	0.63	0.37	0.32
35	2	0.00	sigmoid	0.63	0.37	0.32
36	2	0.00	linear	0.63	0.37	0.32
37	2	0.00	poly	0.63	0.37	0.32
38	2	0.00	rbf	0.63	0.37	0.32
39	2	0.00	sigmoid	0.63	0.37	0.32
40	3	1.00	linear	0.94	0.74	0.79
41	3	1.00	poly	0.80	0.67	0.76
42	3	1.00	rbf	0.91	0.70	0.77
43	3	1.00	sigmoid	0.83	0.63	0.72
44	3	0.50	linear	0.94	0.74	0.79
45	3	0.50	poly	0.74	0.67	0.77
46	3	0.50	rbf	0.87	0.56	0.68
47	3	0.50	sigmoid	0.80	0.59	0.70

48	3	0.25	linear	0.93	0.74	0.79
49	3	0.25	poly	0.70	0.67	0.77
50	3	0.25	rbf	0.76	0.52	0.67
51	3	0.25	sigmoid	0.72	0.52	0.67
52	3	0.10	linear	0.93	0.70	0.77
53	3	0.10	poly	0.69	0.67	0.77
54	3	0.10	rbf	0.76	0.52	0.67
55	3	0.10	sigmoid	0.72	0.52	0.67
56	3	0.05	linear	0.89	0.63	0.72
57	3	0.05	poly	0.67	0.67	0.77
58	3	0.05	rbf	0.76	0.52	0.67
59	3	0.05	sigmoid	0.72	0.52	0.67
60	3	0.03	linear	0.78	0.52	0.67
61	3	0.03	poly	0.67	0.67	0.77
62	3	0.03	rbf	0.76	0.52	0.67
63	3	0.03	sigmoid	0.72	0.52	0.67
64	3	0.01	linear	0.76	0.52	0.67
65	3	0.01	poly	0.63	0.59	0.73
66	3	0.01	rbf	0.76	0.52	0.67
67	3	0.01	sigmoid	0.72	0.52	0.67
68	3	0.01	linear	0.76	0.52	0.67
69	3	0.01	poly	0.63	0.59	0.73
70	3	0.01	rbf	0.76	0.52	0.67
71	3	0.01	sigmoid	0.72	0.52	0.67
72	3	0.00	linear	0.76	0.52	0.67
73	3	0.00	poly	0.63	0.59	0.73
74	3	0.00	rbf	0.76	0.52	0.67
75	3	0.00	sigmoid	0.72	0.52	0.67
76	3	0.00	linear	0.76	0.52	0.67
77	3	0.00	poly	0.63	0.59	0.73
78	3	0.00	rbf	0.76	0.52	0.67
79	3	0.00	sigmoid	0.72	0.52	0.67
80	4	1.00	linear	0.95	0.77	0.81
81	4	1.00	poly	0.85	0.59	0.69
82	4	1.00	rbf	0.93	0.70	0.76
83	4	1.00	sigmoid	0.81	0.66	0.74
84	4	0.50	linear	0.95	0.77	0.81
85	4	0.50	poly	0.75	0.66	0.74
86	4	0.50	rbf	0.88	0.59	0.69
87	4	0.50	sigmoid	0.81	0.55	0.67
88	4	0.25	linear	0.90	0.74	0.79
89	4	0.25	poly	0.78	0.62	0.72
90	4	0.25	rbf	0.62	0.33	0.46
91	4	0.25	sigmoid	0.62	0.33	0.46
92	4	0.10	linear	0.89	0.70	0.76
93	4	0.10	poly	0.69	0.41	0.49
94	4	0.10	rbf	0.62	0.33	0.46

95	4	0.10	sigmoid	0.62	0.33	0.46
96	4	0.05	linear	0.88	0.59	0.69
97	4	0.05	poly	0.67	0.41	0.49
98	4	0.05	rbf	0.62	0.33	0.46
99	4	0.05	sigmoid	0.62	0.33	0.46
100	4	0.03	linear	0.72	0.40	0.57
101	4	0.03	poly	0.67	0.41	0.49
102	4	0.03	rbf	0.62	0.33	0.46
103	4	0.03	sigmoid	0.62	0.33	0.46
104	4	0.01	linear	0.62	0.33	0.46
105	4	0.01	poly	0.62	0.33	0.46
106	4	0.01	rbf	0.62	0.33	0.46
107	4	0.01	sigmoid	0.62	0.33	0.46
108	4	0.01	linear	0.62	0.33	0.46
109	4	0.01	poly	0.62	0.33	0.46
110	4	0.01	rbf	0.62	0.33	0.46
111	4	0.01	sigmoid	0.62	0.33	0.46
112	4	0.00	linear	0.62	0.33	0.46
113	4	0.00	poly	0.62	0.33	0.46
114	4	0.00	rbf	0.62	0.33	0.46
115	4	0.00	sigmoid	0.62	0.33	0.46
116	4	0.00	linear	0.62	0.33	0.46
117	4	0.00	poly	0.62	0.33	0.46
118	4	0.00	rbf	0.62	0.33	0.46
119	4	0.00	sigmoid	0.62	0.33	0.46
120	5	1.00	linear	0.94	0.79	0.83
121	5	1.00	poly	0.84	0.63	0.73
122	5	1.00	rbf	0.93	0.71	0.77
123	5	1.00	sigmoid	0.82	0.64	0.72
124	5	0.50	linear	0.95	0.75	0.77
125	5	0.50	poly	0.72	0.66	0.75
126	5	0.50	rbf	0.87	0.68	0.75
127	5	0.50	sigmoid	0.81	0.64	0.72
128	5	0.25	linear	0.90	0.75	0.80
129	5	0.25	poly	0.70	0.66	0.75
130	5	0.25	rbf	0.66	0.42	0.57
131	5	0.25	sigmoid	0.62	0.42	0.57
132	5	0.10	linear	0.87	0.67	0.74
133	5	0.10	poly	0.69	0.66	0.75
134	5	0.10	rbf	0.60	0.39	0.51
135	5	0.10	sigmoid	0.60	0.39	0.51
136	5	0.05	linear	0.86	0.64	0.72
137	5	0.05	poly	0.68	0.66	0.75
138	5	0.05	rbf	0.60	0.39	0.51
139	5	0.05	sigmoid	0.60	0.39	0.51
140	5	0.03	linear	0.77	0.53	0.67
141	5	0.03	poly	0.66	0.46	0.55

142	5	0.03	rbf	0.60	0.39	0.51
143	5	0.03	sigmoid	0.60	0.39	0.51
144	5	0.01	linear	0.60	0.39	0.51
145	5	0.01	poly	0.60	0.39	0.51
146	5	0.01	rbf	0.60	0.39	0.51
147	5	0.01	sigmoid	0.60	0.39	0.51
148	5	0.01	linear	0.60	0.39	0.51
149	5	0.01	poly	0.60	0.39	0.51
150	5	0.01	rbf	0.60	0.39	0.51
151	5	0.01	sigmoid	0.60	0.39	0.51
152	5	0.00	linear	0.60	0.39	0.51
153	5	0.00	poly	0.60	0.39	0.51
154	5	0.00	rbf	0.60	0.39	0.51
155	5	0.00	sigmoid	0.60	0.39	0.51
156	5	0.00	linear	0.60	0.39	0.51
157	5	0.00	poly	0.60	0.39	0.51
158	5	0.00	rbf	0.60	0.39	0.51
159	5	0.00	sigmoid	0.60	0.39	0.51

	Jaccard Score
0	0.72
1	0.61
2	0.62
3	0.57
4	0.72
5	0.40
6	0.48
7	0.55
8	0.72
9	0.25
10	0.23
11	0.23
12	0.62
13	0.25
14	0.23
15	0.23
16	0.55
17	0.25
18	0.23
19	0.23
20	0.23
21	0.25
22	0.23
23	0.23
24	0.23
25	0.23
26	0.23

27	0.23
28	0.23
29	0.23
30	0.23
31	0.23
32	0.23
33	0.23
34	0.23
35	0.23
36	0.23
37	0.23
38	0.23
39	0.23
40	0.67
41	0.61
42	0.63
43	0.57
44	0.67
45	0.63
46	0.52
47	0.54
48	0.67
49	0.63
50	0.50
51	0.50
52	0.63
53	0.63
54	0.50
55	0.50
56	0.57
57	0.63
58	0.50
59	0.50
60	0.50
61	0.63
62	0.50
63	0.50
64	0.50
65	0.59
66	0.50
67	0.50
68	0.50
69	0.59
70	0.50
71	0.50
72	0.50
73	0.59

74	0.50
75	0.50
76	0.50
77	0.59
78	0.50
79	0.50
80	0.71
81	0.54
82	0.62
83	0.62
84	0.71
85	0.62
86	0.53
87	0.51
88	0.66
89	0.58
90	0.33
91	0.33
92	0.62
93	0.37
94	0.33
95	0.33
96	0.53
97	0.37
98	0.33
99	0.33
100	0.40
101	0.37
102	0.33
103	0.33
104	0.33
105	0.33
106	0.33
107	0.33
108	0.33
109	0.33
110	0.33
111	0.33
112	0.33
113	0.33
114	0.33
115	0.33
116	0.33
117	0.33
118	0.33
119	0.33
120	0.73

```
121           0.58
122           0.65
123           0.58
124           0.68
125           0.63
126           0.63
127           0.58
128           0.68
129           0.63
130           0.42
131           0.42
132           0.60
133           0.63
134           0.39
135           0.39
136           0.58
137           0.63
138           0.39
139           0.39
140           0.51
141           0.43
142           0.39
143           0.39
144           0.39
145           0.39
146           0.39
147           0.39
148           0.39
149           0.39
150           0.39
151           0.39
152           0.39
153           0.39
154           0.39
155           0.39
156           0.39
157           0.39
158           0.39
159           0.39
```

[40]:

```
clf_svm_eval_df.nlargest(3, 'Test-set Accuracy')
```

[40]:

```
   k     c  kernel  Train-set Accuracy  Test-set Accuracy  F1 Score  \
0  2  1.00  linear                0.96               0.82      0.84
4  2  0.50  linear                0.96               0.82      0.84
8  2  0.25  linear                0.93               0.82      0.84
```

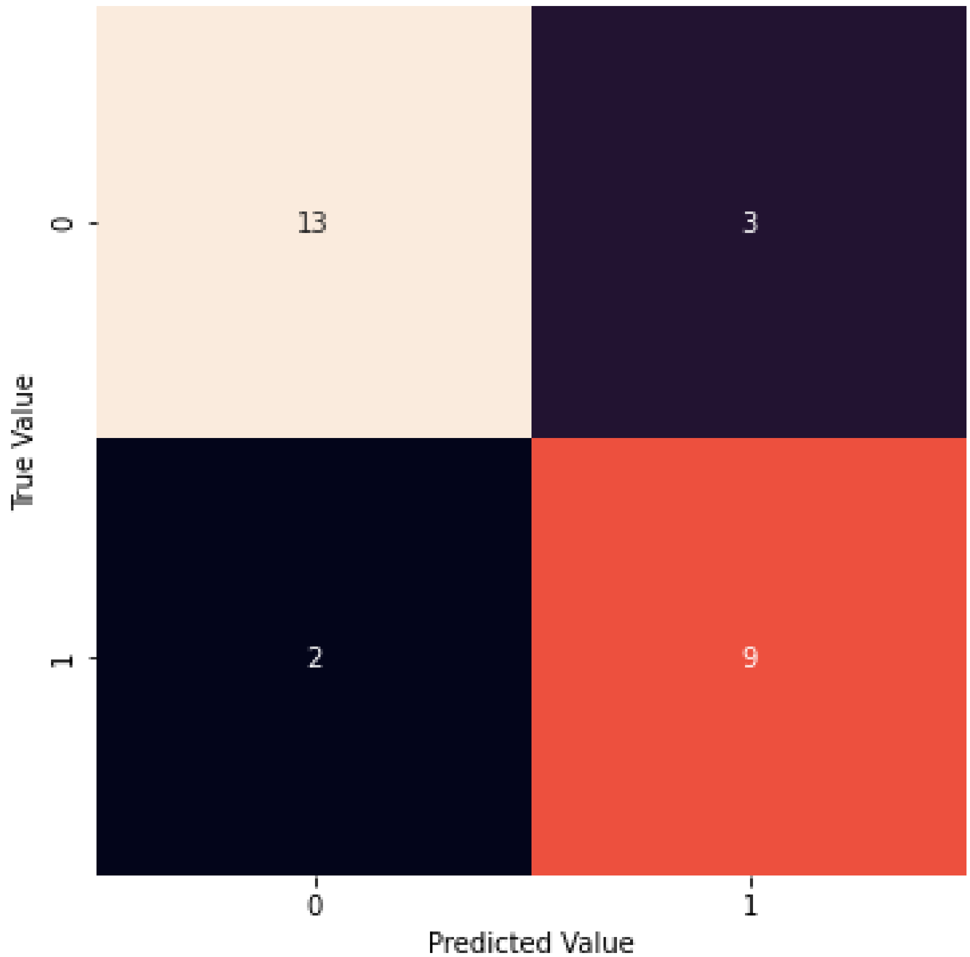
```
   Jaccard Score
0           0.72
4           0.72
8           0.72
```

[41]:
```
kf = KFold(n_splits = 2)
temp_lst = []
clf_svm = SVC(C=1, kernel='linear')
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]
    clf_svm.fit(X_train, y_train)
    y_hat = clf_svm.predict(X_test)
    temp_lst2 = []
    temp_lst2.append(metrics.accuracy_score(y_train, clf_svm.predict(X_train)))
    temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
    temp_lst2.append(metrics.f1_score(y_test, y_hat))
    temp_lst2.append(metrics.jaccard_score(y_test, y_hat))
    temp_lst2.append(y_test)
    temp_lst2.append(y_hat)
    temp_lst2.append(X_test)
    temp_lst.append(temp_lst2)
```

[42]:
```
temp_lst_ytest, temp_lst_yhat, temp_lst_xtest = [], [], []
for row in temp_lst:
    for i in row[4]:
        temp_lst_ytest.append(i)
    for j in row[5]:
        temp_lst_yhat.append(j)
    for k in row[6]:
        temp_lst_xtest.append(k)

cnf_ytest = np.array(temp_lst_ytest)
cnf_yhat = np.array(temp_lst_yhat)
cnf_xtest = np.array(temp_lst_xtest)
print(metrics.classification_report(cnf_ytest, cnf_yhat))
```

```
              precision    recall  f1-score   support

           0       0.75      0.82      0.78        11
           1       0.87      0.81      0.84        16

    accuracy                           0.81        27
   macro avg       0.81      0.82      0.81        27
weighted avg       0.82      0.81      0.82        27
```

[43]:
```
cnf_matrix = confusion_matrix(cnf_ytest, cnf_yhat, labels=[1,0])
plt.figure(figsize=(8,6))
sns.heatmap(cnf_matrix, square=True, annot=True, cbar=False)
plt.xlabel('Predicted Value')
plt.ylabel('True Value')
plt.show()
```



Influencers

[44]:
```
temp_lst = []
for i in range(2, 6):
    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:
```

```
                clf_svm = SVC(C=c, kernel=kernel_type)
                clf_svm.fit(X_train, y_train)
                y_hat = clf_svm.predict(X_test)
                temp_lst2 = []
                temp_lst2.append(i)
                temp_lst2.append(c)
                temp_lst2.append(kernel_type)
                temp_lst2.append(metrics.accuracy_score(y_train, clf_svm.
 ↪predict(X_train)))
                temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
                temp_lst2.append(metrics.f1_score(y_test, y_hat,␣
 ↪average='micro'))
                temp_lst2.append(metrics.jaccard_score(y_test, y_hat,␣
 ↪average='micro'))
                temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst,
                       columns=['k', 'c', 'kernel', 'Train-set Accuracy',␣
 ↪'Test-set Accuracy', 'F1 Score', 'Jaccard Score'])
temp_lst = []
for k in range(2, 6):
    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-set␣
 ↪Accuracy']), 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-set␣
 ↪Accuracy']), decimals=4))
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['c'] == c) & (temp_df['kernel'] == kernel_type)]['F1 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)]['Jaccard Score']),␣
 ↪decimals=4))
            temp_lst.append(temp_lst2)

clf_svm_eval_df = pd.DataFrame(temp_lst,
                              columns=['k', 'c', 'kernel', 'Train-set␣
 ↪Accuracy', 'Test-set Accuracy', 'F1 Score', 'Jaccard Score'])
clf_svm_eval_df
```

[44]:

	k	c	kernel	Train-set Accuracy	Test-set Accuracy	F1 Score	\
0	2	1.00	linear	0.89	0.68	0.68	
1	2	1.00	poly	0.85	0.55	0.55	
2	2	1.00	rbf	0.83	0.58	0.58	
3	2	1.00	sigmoid	0.74	0.67	0.67	
4	2	0.50	linear	0.88	0.58	0.58	
5	2	0.50	poly	0.75	0.53	0.53	
6	2	0.50	rbf	0.80	0.52	0.52	
7	2	0.50	sigmoid	0.70	0.66	0.66	
8	2	0.25	linear	0.83	0.58	0.58	
9	2	0.25	poly	0.65	0.50	0.50	
10	2	0.25	rbf	0.71	0.43	0.43	
11	2	0.25	sigmoid	0.54	0.50	0.50	
12	2	0.10	linear	0.76	0.61	0.61	
13	2	0.10	poly	0.61	0.48	0.48	
14	2	0.10	rbf	0.48	0.42	0.42	
15	2	0.10	sigmoid	0.53	0.42	0.42	
16	2	0.05	linear	0.76	0.61	0.61	
17	2	0.05	poly	0.56	0.44	0.44	
18	2	0.05	rbf	0.42	0.42	0.42	
19	2	0.05	sigmoid	0.42	0.42	0.42	
20	2	0.03	linear	0.74	0.53	0.53	
21	2	0.03	poly	0.44	0.42	0.42	
22	2	0.03	rbf	0.42	0.42	0.42	
23	2	0.03	sigmoid	0.42	0.42	0.42	
24	2	0.01	linear	0.57	0.42	0.42	
25	2	0.01	poly	0.44	0.42	0.42	
26	2	0.01	rbf	0.42	0.42	0.42	
27	2	0.01	sigmoid	0.42	0.42	0.42	
28	2	0.01	linear	0.53	0.42	0.42	
29	2	0.01	poly	0.44	0.42	0.42	
30	2	0.01	rbf	0.42	0.42	0.42	
31	2	0.01	sigmoid	0.42	0.42	0.42	
32	2	0.00	linear	0.44	0.42	0.42	
33	2	0.00	poly	0.44	0.42	0.42	
34	2	0.00	rbf	0.42	0.42	0.42	
35	2	0.00	sigmoid	0.42	0.42	0.42	
36	2	0.00	linear	0.42	0.42	0.42	
37	2	0.00	poly	0.42	0.42	0.42	
38	2	0.00	rbf	0.42	0.42	0.42	
39	2	0.00	sigmoid	0.42	0.42	0.42	
40	3	1.00	linear	0.93	0.49	0.49	
41	3	1.00	poly	0.81	0.47	0.47	
42	3	1.00	rbf	0.84	0.38	0.38	
43	3	1.00	sigmoid	0.80	0.44	0.44	
44	3	0.50	linear	0.91	0.51	0.51	
45	3	0.50	poly	0.73	0.32	0.32	

46	3	0.50	rbf	0.81	0.38	0.38
47	3	0.50	sigmoid	0.73	0.37	0.37
48	3	0.25	linear	0.85	0.51	0.51
49	3	0.25	poly	0.70	0.32	0.32
50	3	0.25	rbf	0.74	0.26	0.26
51	3	0.25	sigmoid	0.67	0.24	0.24
52	3	0.10	linear	0.82	0.42	0.42
53	3	0.10	poly	0.59	0.26	0.26
54	3	0.10	rbf	0.50	0.21	0.21
55	3	0.10	sigmoid	0.51	0.21	0.21
56	3	0.05	linear	0.79	0.42	0.42
57	3	0.05	poly	0.57	0.23	0.23
58	3	0.05	rbf	0.50	0.21	0.21
59	3	0.05	sigmoid	0.50	0.21	0.21
60	3	0.03	linear	0.74	0.35	0.35
61	3	0.03	poly	0.50	0.21	0.21
62	3	0.03	rbf	0.50	0.21	0.21
63	3	0.03	sigmoid	0.50	0.21	0.21
64	3	0.01	linear	0.66	0.25	0.25
65	3	0.01	poly	0.50	0.21	0.21
66	3	0.01	rbf	0.50	0.21	0.21
67	3	0.01	sigmoid	0.50	0.21	0.21
68	3	0.01	linear	0.55	0.21	0.21
69	3	0.01	poly	0.50	0.21	0.21
70	3	0.01	rbf	0.50	0.21	0.21
71	3	0.01	sigmoid	0.50	0.21	0.21
72	3	0.00	linear	0.50	0.21	0.21
73	3	0.00	poly	0.50	0.21	0.21
74	3	0.00	rbf	0.50	0.21	0.21
75	3	0.00	sigmoid	0.50	0.21	0.21
76	3	0.00	linear	0.50	0.21	0.21
77	3	0.00	poly	0.50	0.21	0.21
78	3	0.00	rbf	0.50	0.21	0.21
79	3	0.00	sigmoid	0.50	0.21	0.21
80	4	1.00	linear	0.92	0.61	0.61
81	4	1.00	poly	0.77	0.48	0.48
82	4	1.00	rbf	0.80	0.33	0.33
83	4	1.00	sigmoid	0.77	0.50	0.50
84	4	0.50	linear	0.88	0.55	0.55
85	4	0.50	poly	0.72	0.45	0.45
86	4	0.50	rbf	0.80	0.35	0.35
87	4	0.50	sigmoid	0.70	0.45	0.45
88	4	0.25	linear	0.83	0.48	0.48
89	4	0.25	poly	0.71	0.48	0.48
90	4	0.25	rbf	0.73	0.34	0.34
91	4	0.25	sigmoid	0.66	0.39	0.39
92	4	0.10	linear	0.78	0.41	0.41

93	4	0.10	poly	0.60	0.43	0.43
94	4	0.10	rbf	0.53	0.20	0.20
95	4	0.10	sigmoid	0.57	0.20	0.20
96	4	0.05	linear	0.76	0.43	0.43
97	4	0.05	poly	0.54	0.29	0.29
98	4	0.05	rbf	0.45	0.28	0.28
99	4	0.05	sigmoid	0.47	0.28	0.28
100	4	0.03	linear	0.72	0.41	0.41
101	4	0.03	poly	0.51	0.28	0.28
102	4	0.03	rbf	0.45	0.28	0.28
103	4	0.03	sigmoid	0.45	0.28	0.28
104	4	0.01	linear	0.66	0.32	0.32
105	4	0.01	poly	0.46	0.28	0.28
106	4	0.01	rbf	0.45	0.28	0.28
107	4	0.01	sigmoid	0.45	0.28	0.28
108	4	0.01	linear	0.59	0.26	0.26
109	4	0.01	poly	0.46	0.28	0.28
110	4	0.01	rbf	0.45	0.28	0.28
111	4	0.01	sigmoid	0.45	0.28	0.28
112	4	0.00	linear	0.50	0.28	0.28
113	4	0.00	poly	0.46	0.28	0.28
114	4	0.00	rbf	0.45	0.28	0.28
115	4	0.00	sigmoid	0.45	0.28	0.28
116	4	0.00	linear	0.45	0.28	0.28
117	4	0.00	poly	0.45	0.28	0.28
118	4	0.00	rbf	0.45	0.28	0.28
119	4	0.00	sigmoid	0.45	0.28	0.28
120	5	1.00	linear	0.93	0.71	0.71
121	5	1.00	poly	0.75	0.54	0.54
122	5	1.00	rbf	0.81	0.41	0.41
123	5	1.00	sigmoid	0.74	0.56	0.56
124	5	0.50	linear	0.89	0.63	0.63
125	5	0.50	poly	0.71	0.46	0.46
126	5	0.50	rbf	0.76	0.42	0.42
127	5	0.50	sigmoid	0.72	0.50	0.50
128	5	0.25	linear	0.83	0.55	0.55
129	5	0.25	poly	0.67	0.50	0.50
130	5	0.25	rbf	0.72	0.34	0.34
131	5	0.25	sigmoid	0.64	0.38	0.38
132	5	0.10	linear	0.77	0.51	0.51
133	5	0.10	poly	0.61	0.44	0.44
134	5	0.10	rbf	0.50	0.24	0.24
135	5	0.10	sigmoid	0.57	0.37	0.37
136	5	0.05	linear	0.75	0.50	0.50
137	5	0.05	poly	0.55	0.42	0.42
138	5	0.05	rbf	0.43	0.24	0.24
139	5	0.05	sigmoid	0.46	0.24	0.24

140	5	0.03	linear	0.71	0.47	0.47
141	5	0.03	poly	0.52	0.32	0.32
142	5	0.03	rbf	0.43	0.24	0.24
143	5	0.03	sigmoid	0.43	0.24	0.24
144	5	0.01	linear	0.63	0.41	0.41
145	5	0.01	poly	0.44	0.24	0.24
146	5	0.01	rbf	0.43	0.24	0.24
147	5	0.01	sigmoid	0.43	0.24	0.24
148	5	0.01	linear	0.57	0.37	0.37
149	5	0.01	poly	0.44	0.24	0.24
150	5	0.01	rbf	0.43	0.24	0.24
151	5	0.01	sigmoid	0.43	0.24	0.24
152	5	0.00	linear	0.50	0.24	0.24
153	5	0.00	poly	0.44	0.24	0.24
154	5	0.00	rbf	0.43	0.24	0.24
155	5	0.00	sigmoid	0.43	0.24	0.24
156	5	0.00	linear	0.43	0.24	0.24
157	5	0.00	poly	0.44	0.24	0.24
158	5	0.00	rbf	0.43	0.24	0.24
159	5	0.00	sigmoid	0.43	0.24	0.24

	Jaccard Score
0	0.51
1	0.38
2	0.41
3	0.50
4	0.41
5	0.36
6	0.35
7	0.49
8	0.41
9	0.34
10	0.28
11	0.33
12	0.44
13	0.32
14	0.27
15	0.27
16	0.44
17	0.28
18	0.27
19	0.27
20	0.36
21	0.27
22	0.27
23	0.27
24	0.27

25	0.27
26	0.27
27	0.27
28	0.27
29	0.27
30	0.27
31	0.27
32	0.27
33	0.27
34	0.27
35	0.27
36	0.27
37	0.27
38	0.27
39	0.27
40	0.34
41	0.31
42	0.24
43	0.29
44	0.36
45	0.20
46	0.24
47	0.23
48	0.35
49	0.20
50	0.15
51	0.14
52	0.27
53	0.16
54	0.12
55	0.12
56	0.27
57	0.13
58	0.12
59	0.12
60	0.22
61	0.12
62	0.12
63	0.12
64	0.14
65	0.12
66	0.12
67	0.12
68	0.12
69	0.12
70	0.12
71	0.12

72	0.12
73	0.12
74	0.12
75	0.12
76	0.12
77	0.12
78	0.12
79	0.12
80	0.45
81	0.33
82	0.20
83	0.33
84	0.40
85	0.32
86	0.22
87	0.31
88	0.33
89	0.35
90	0.24
91	0.25
92	0.26
93	0.30
94	0.11
95	0.11
96	0.28
97	0.17
98	0.17
99	0.17
100	0.29
101	0.17
102	0.17
103	0.17
104	0.21
105	0.17
106	0.17
107	0.17
108	0.15
109	0.17
110	0.17
111	0.17
112	0.17
113	0.17
114	0.17
115	0.17
116	0.17
117	0.17
118	0.17

```
119            0.17
120            0.59
121            0.42
122            0.29
123            0.43
124            0.49
125            0.33
126            0.33
127            0.37
128            0.40
129            0.36
130            0.24
131            0.25
132            0.40
133            0.30
134            0.13
135            0.24
136            0.39
137            0.29
138            0.13
139            0.13
140            0.37
141            0.20
142            0.13
143            0.13
144            0.27
145            0.13
146            0.13
147            0.13
148            0.24
149            0.13
150            0.13
151            0.13
152            0.13
153            0.13
154            0.13
155            0.13
156            0.13
157            0.13
158            0.13
159            0.13
```

[45]:
```
clf_svm_eval_df.nlargest(3, 'Test-set Accuracy')
```

[45]:
```
     k    c   kernel  Train-set Accuracy  Test-set Accuracy  F1 Score  \
120  5 1.00   linear                0.93               0.71      0.71
0    2 1.00   linear                0.89               0.68      0.68
```

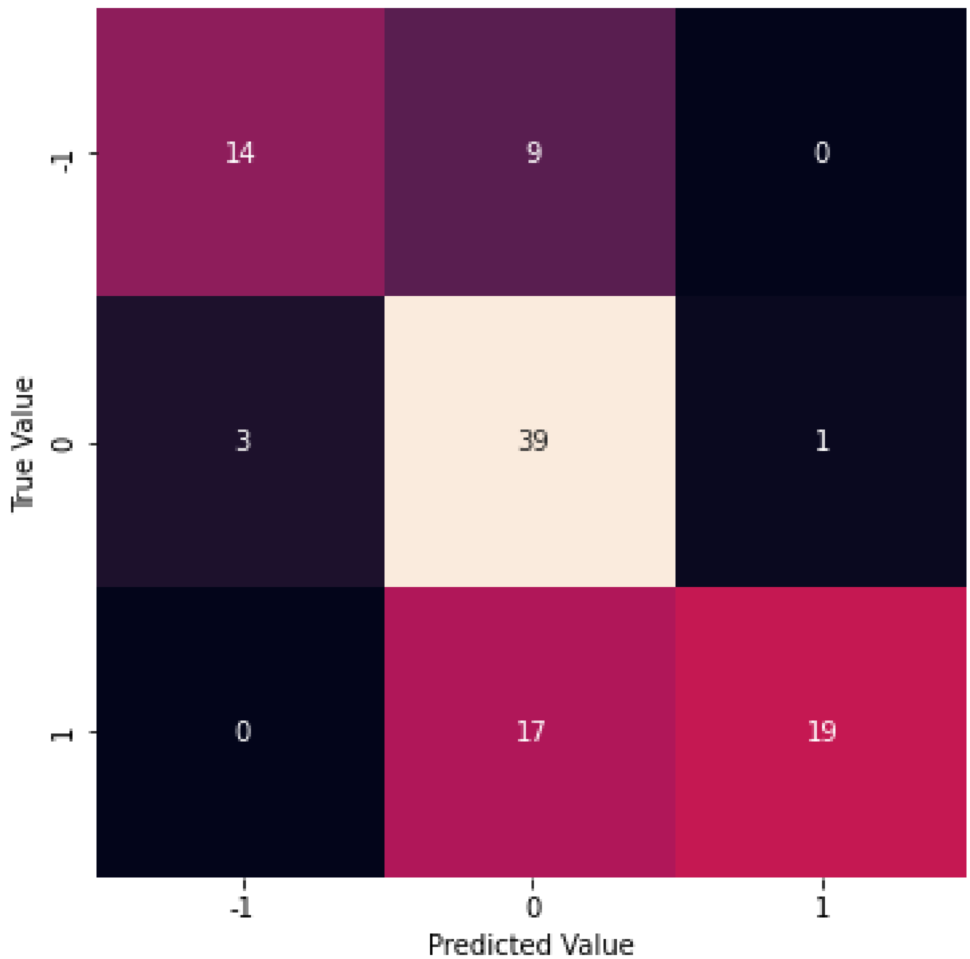
```
3    2 1.00  sigmoid                0.74              0.67      0.67

     Jaccard Score
120           0.59
0             0.51
3             0.50
```

[46]:
```
kf = KFold(n_splits = 5)
temp_lst = []
clf_svm = SVC(C=1, kernel='linear')
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]
    clf_svm.fit(X_train, y_train)
    y_hat = clf_svm.predict(X_test)
    temp_lst2 = []
    temp_lst2.append(metrics.accuracy_score(y_train, clf_svm.predict(X_train)))
    temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
    temp_lst2.append(metrics.f1_score(y_test, y_hat, average='micro'))
    temp_lst2.append(metrics.jaccard_score(y_test, y_hat, average='micro'))
    temp_lst2.append(y_test)
    temp_lst2.append(y_hat)
    temp_lst2.append(X_test)
    temp_lst.append(temp_lst2)
```

[47]:
```
temp_lst_ytest, temp_lst_yhat, temp_lst_xtest = [], [], []
for row in temp_lst:
    for i in row[4]:
        temp_lst_ytest.append(i)
    for j in row[5]:
        temp_lst_yhat.append(j)
    for k in row[6]:
        temp_lst_xtest.append(k)

cnf_ytest = np.array(temp_lst_ytest)
cnf_yhat = np.array(temp_lst_yhat)
cnf_xtest = np.array(temp_lst_xtest)
print(metrics.classification_report(cnf_ytest, cnf_yhat))
```

```
              precision    recall  f1-score   support

          -1       0.82      0.61      0.70        23
           0       0.60      0.91      0.72        43
           1       0.95      0.53      0.68        36

    accuracy                           0.71       102
   macro avg       0.79      0.68      0.70       102
weighted avg       0.77      0.71      0.70       102
```

[48]:
```python
cnf_matrix = confusion_matrix(cnf_ytest, cnf_yhat)
plt.figure(figsize=(8,6))
sns.heatmap(cnf_matrix, square=True, annot=True, cbar=False)
plt.xticks([.5, 1.5, 2.5], [-1, 0, 1])
plt.yticks([.5, 1.5, 2.5], [-1, 0, 1])
plt.xlabel('Predicted Value')
plt.ylabel('True Value')
plt.show()
```



Leaders Post

[49]:
```python
temp_lst = []
for i in range(2, 6):
    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:
                clf_svm = SVC(C=c, kernel=kernel_type)
                clf_svm.fit(X_train, y_train)
                y_hat = clf_svm.predict(X_test)
                temp_lst2 = []
                temp_lst2.append(i)
                temp_lst2.append(c)
                temp_lst2.append(kernel_type)
                temp_lst2.append(metrics.accuracy_score(y_train, clf_svm.
↪predict(X_train)))
                temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
                temp_lst2.append(metrics.f1_score(y_test, y_hat,␣
↪average='micro'))
                temp_lst2.append(metrics.jaccard_score(y_test, y_hat,␣
↪average='micro'))
                temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst,
                       columns=['k', 'c', 'kernel', 'Train-set Accuracy',␣
↪'Test-set Accuracy', 'F1 Score', 'Jaccard Score'])
temp_lst = []
for k in range(2, 6):
    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-set␣
↪Accuracy']), 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-set␣
↪Accuracy']), decimals=4))
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['c'] == c) & (temp_df['kernel'] == kernel_type)]['F1 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)]['Jaccard Score']),␣
↪decimals=4))
            temp_lst.append(temp_lst2)
```

```
clf_svm_eval_df = pd.DataFrame(temp_lst,
                              columns=['k', 'c', 'kernel', 'Train-set␣
↪Accuracy', 'Test-set Accuracy', 'F1 Score', 'Jaccard Score'])
clf_svm_eval_df
```

[49]:

```
     k     c   kernel  Train-set Accuracy  Test-set Accuracy  F1 Score  \
0    2  1.00   linear                1.00               0.35      0.35
1    2  1.00     poly                0.78               0.57      0.57
2    2  1.00      rbf                1.00               0.45      0.45
3    2  1.00  sigmoid                0.78               0.45      0.45
4    2  0.50   linear                1.00               0.35      0.35
5    2  0.50     poly                0.78               0.57      0.57
6    2  0.50      rbf                0.78               0.45      0.45
7    2  0.50  sigmoid                0.78               0.45      0.45
8    2  0.25   linear                1.00               0.35      0.35
9    2  0.25     poly                0.68               0.57      0.57
10   2  0.25      rbf                0.78               0.45      0.45
11   2  0.25  sigmoid                0.78               0.45      0.45
12   2  0.10   linear                0.88               0.45      0.45
13   2  0.10     poly                0.68               0.57      0.57
14   2  0.10      rbf                0.78               0.45      0.45
15   2  0.10  sigmoid                0.78               0.45      0.45
16   2  0.05   linear                0.88               0.45      0.45
17   2  0.05     poly                0.68               0.57      0.57
18   2  0.05      rbf                0.78               0.45      0.45
19   2  0.05  sigmoid                0.78               0.45      0.45
20   2  0.03   linear                0.78               0.45      0.45
21   2  0.03     poly                0.68               0.57      0.57
22   2  0.03      rbf                0.78               0.45      0.45
23   2  0.03  sigmoid                0.78               0.45      0.45
24   2  0.01   linear                0.78               0.45      0.45
25   2  0.01     poly                0.68               0.57      0.57
26   2  0.01      rbf                0.78               0.45      0.45
27   2  0.01  sigmoid                0.78               0.45      0.45
28   2  0.01   linear                0.78               0.45      0.45
29   2  0.01     poly                0.68               0.57      0.57
30   2  0.01      rbf                0.78               0.45      0.45
31   2  0.01  sigmoid                0.78               0.45      0.45
32   2  0.00   linear                0.78               0.45      0.45
33   2  0.00     poly                0.68               0.57      0.57
34   2  0.00      rbf                0.78               0.45      0.45
35   2  0.00  sigmoid                0.78               0.45      0.45
36   2  0.00   linear                0.78               0.45      0.45
37   2  0.00     poly                0.68               0.57      0.57
38   2  0.00      rbf                0.78               0.45      0.45
39   2  0.00  sigmoid                0.78               0.45      0.45
```

40	3	1.00	linear	1.00	0.56	0.56
41	3	1.00	poly	0.83	0.44	0.44
42	3	1.00	rbf	0.83	0.44	0.44
43	3	1.00	sigmoid	0.83	0.44	0.44
44	3	0.50	linear	0.89	0.56	0.56
45	3	0.50	poly	0.83	0.44	0.44
46	3	0.50	rbf	0.72	0.44	0.44
47	3	0.50	sigmoid	0.72	0.44	0.44
48	3	0.25	linear	0.83	0.56	0.56
49	3	0.25	poly	0.78	0.44	0.44
50	3	0.25	rbf	0.72	0.44	0.44
51	3	0.25	sigmoid	0.72	0.44	0.44
52	3	0.10	linear	0.83	0.56	0.56
53	3	0.10	poly	0.72	0.44	0.44
54	3	0.10	rbf	0.72	0.44	0.44
55	3	0.10	sigmoid	0.72	0.44	0.44
56	3	0.05	linear	0.83	0.44	0.44
57	3	0.05	poly	0.67	0.44	0.44
58	3	0.05	rbf	0.72	0.44	0.44
59	3	0.05	sigmoid	0.72	0.44	0.44
60	3	0.03	linear	0.78	0.44	0.44
61	3	0.03	poly	0.67	0.44	0.44
62	3	0.03	rbf	0.72	0.44	0.44
63	3	0.03	sigmoid	0.72	0.44	0.44
64	3	0.01	linear	0.72	0.44	0.44
65	3	0.01	poly	0.67	0.44	0.44
66	3	0.01	rbf	0.72	0.44	0.44
67	3	0.01	sigmoid	0.72	0.44	0.44
68	3	0.01	linear	0.72	0.44	0.44
69	3	0.01	poly	0.67	0.44	0.44
70	3	0.01	rbf	0.72	0.44	0.44
71	3	0.01	sigmoid	0.72	0.44	0.44
72	3	0.00	linear	0.72	0.44	0.44
73	3	0.00	poly	0.67	0.44	0.44
74	3	0.00	rbf	0.72	0.44	0.44
75	3	0.00	sigmoid	0.72	0.44	0.44
76	3	0.00	linear	0.72	0.44	0.44
77	3	0.00	poly	0.67	0.44	0.44
78	3	0.00	rbf	0.72	0.44	0.44
79	3	0.00	sigmoid	0.72	0.44	0.44
80	4	1.00	linear	1.00	0.42	0.42
81	4	1.00	poly	0.82	0.58	0.58
82	4	1.00	rbf	0.82	0.46	0.46
83	4	1.00	sigmoid	0.82	0.46	0.46
84	4	0.50	linear	0.89	0.42	0.42
85	4	0.50	poly	0.78	0.58	0.58
86	4	0.50	rbf	0.60	0.58	0.58

87	4	0.50	sigmoid	0.74	0.46	0.46
88	4	0.25	linear	0.85	0.42	0.42
89	4	0.25	poly	0.71	0.58	0.58
90	4	0.25	rbf	0.56	0.58	0.58
91	4	0.25	sigmoid	0.56	0.58	0.58
92	4	0.10	linear	0.82	0.54	0.54
93	4	0.10	poly	0.63	0.58	0.58
94	4	0.10	rbf	0.56	0.58	0.58
95	4	0.10	sigmoid	0.56	0.58	0.58
96	4	0.05	linear	0.82	0.46	0.46
97	4	0.05	poly	0.56	0.58	0.58
98	4	0.05	rbf	0.56	0.58	0.58
99	4	0.05	sigmoid	0.56	0.58	0.58
100	4	0.03	linear	0.70	0.58	0.58
101	4	0.03	poly	0.56	0.58	0.58
102	4	0.03	rbf	0.56	0.58	0.58
103	4	0.03	sigmoid	0.56	0.58	0.58
104	4	0.01	linear	0.56	0.58	0.58
105	4	0.01	poly	0.56	0.58	0.58
106	4	0.01	rbf	0.56	0.58	0.58
107	4	0.01	sigmoid	0.56	0.58	0.58
108	4	0.01	linear	0.56	0.58	0.58
109	4	0.01	poly	0.56	0.58	0.58
110	4	0.01	rbf	0.56	0.58	0.58
111	4	0.01	sigmoid	0.56	0.58	0.58
112	4	0.00	linear	0.56	0.58	0.58
113	4	0.00	poly	0.56	0.58	0.58
114	4	0.00	rbf	0.56	0.58	0.58
115	4	0.00	sigmoid	0.56	0.58	0.58
116	4	0.00	linear	0.56	0.58	0.58
117	4	0.00	poly	0.56	0.58	0.58
118	4	0.00	rbf	0.56	0.58	0.58
119	4	0.00	sigmoid	0.56	0.58	0.58
120	5	1.00	linear	1.00	0.50	0.50
121	5	1.00	poly	0.81	0.50	0.50
122	5	1.00	rbf	0.81	0.50	0.50
123	5	1.00	sigmoid	0.81	0.50	0.50
124	5	0.50	linear	0.89	0.50	0.50
125	5	0.50	poly	0.78	0.60	0.60
126	5	0.50	rbf	0.61	0.60	0.60
127	5	0.50	sigmoid	0.72	0.50	0.50
128	5	0.25	linear	0.81	0.50	0.50
129	5	0.25	poly	0.67	0.60	0.60
130	5	0.25	rbf	0.56	0.60	0.60
131	5	0.25	sigmoid	0.61	0.60	0.60
132	5	0.10	linear	0.81	0.50	0.50
133	5	0.10	poly	0.67	0.60	0.60

134	5	0.10	rbf	0.56	0.60	0.60
135	5	0.10	sigmoid	0.56	0.60	0.60
136	5	0.05	linear	0.81	0.50	0.50
137	5	0.05	poly	0.56	0.60	0.60
138	5	0.05	rbf	0.56	0.60	0.60
139	5	0.05	sigmoid	0.56	0.60	0.60
140	5	0.03	linear	0.72	0.50	0.50
141	5	0.03	poly	0.56	0.60	0.60
142	5	0.03	rbf	0.56	0.60	0.60
143	5	0.03	sigmoid	0.56	0.60	0.60
144	5	0.01	linear	0.56	0.60	0.60
145	5	0.01	poly	0.56	0.60	0.60
146	5	0.01	rbf	0.56	0.60	0.60
147	5	0.01	sigmoid	0.56	0.60	0.60
148	5	0.01	linear	0.56	0.60	0.60
149	5	0.01	poly	0.56	0.60	0.60
150	5	0.01	rbf	0.56	0.60	0.60
151	5	0.01	sigmoid	0.56	0.60	0.60
152	5	0.00	linear	0.56	0.60	0.60
153	5	0.00	poly	0.56	0.60	0.60
154	5	0.00	rbf	0.56	0.60	0.60
155	5	0.00	sigmoid	0.56	0.60	0.60
156	5	0.00	linear	0.56	0.60	0.60
157	5	0.00	poly	0.56	0.60	0.60
158	5	0.00	rbf	0.56	0.60	0.60
159	5	0.00	sigmoid	0.56	0.60	0.60

	Jaccard Score
0	0.22
1	0.42
2	0.29
3	0.29
4	0.22
5	0.42
6	0.29
7	0.29
8	0.22
9	0.42
10	0.29
11	0.29
12	0.29
13	0.42
14	0.29
15	0.29
16	0.29
17	0.42
18	0.29

19	0.29
20	0.29
21	0.42
22	0.29
23	0.29
24	0.29
25	0.42
26	0.29
27	0.29
28	0.29
29	0.42
30	0.29
31	0.29
32	0.29
33	0.42
34	0.29
35	0.29
36	0.29
37	0.42
38	0.29
39	0.29
40	0.40
41	0.30
42	0.30
43	0.30
44	0.40
45	0.30
46	0.30
47	0.30
48	0.40
49	0.30
50	0.30
51	0.30
52	0.40
53	0.30
54	0.30
55	0.30
56	0.30
57	0.30
58	0.30
59	0.30
60	0.30
61	0.30
62	0.30
63	0.30
64	0.30
65	0.30

66	0.30
67	0.30
68	0.30
69	0.30
70	0.30
71	0.30
72	0.30
73	0.30
74	0.30
75	0.30
76	0.30
77	0.30
78	0.30
79	0.30
80	0.29
81	0.47
82	0.30
83	0.30
84	0.29
85	0.47
86	0.47
87	0.30
88	0.29
89	0.47
90	0.47
91	0.47
92	0.38
93	0.47
94	0.47
95	0.47
96	0.30
97	0.47
98	0.47
99	0.47
100	0.47
101	0.47
102	0.47
103	0.47
104	0.47
105	0.47
106	0.47
107	0.47
108	0.47
109	0.47
110	0.47
111	0.47
112	0.47

113	0.47
114	0.47
115	0.47
116	0.47
117	0.47
118	0.47
119	0.47
120	0.40
121	0.40
122	0.40
123	0.40
124	0.40
125	0.53
126	0.53
127	0.40
128	0.40
129	0.53
130	0.53
131	0.53
132	0.40
133	0.53
134	0.53
135	0.53
136	0.40
137	0.53
138	0.53
139	0.53
140	0.40
141	0.53
142	0.53
143	0.53
144	0.53
145	0.53
146	0.53
147	0.53
148	0.53
149	0.53
150	0.53
151	0.53
152	0.53
153	0.53
154	0.53
155	0.53
156	0.53
157	0.53
158	0.53
159	0.53

[50]:
```
clf_svm_eval_df.nlargest(3, 'Test-set Accuracy')
```

[50]:
```
     k     c kernel  Train-set Accuracy  Test-set Accuracy  F1 Score  \
125  5  0.50   poly                0.78               0.60      0.60
126  5  0.50    rbf                0.61               0.60      0.60
129  5  0.25   poly                0.67               0.60      0.60

     Jaccard Score
125           0.53
126           0.53
129           0.53
```

[51]:
```
kf = KFold(n_splits = 5)
temp_lst = []
clf_svm = SVC(C=1, kernel='linear')
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]
    clf_svm.fit(X_train, y_train)
    y_hat = clf_svm.predict(X_test)
    temp_lst2 = []
    temp_lst2.append(metrics.accuracy_score(y_train, clf_svm.predict(X_train)))
    temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
    temp_lst2.append(metrics.f1_score(y_test, y_hat, average='micro'))
    temp_lst2.append(metrics.jaccard_score(y_test, y_hat, average='micro'))
    temp_lst2.append(y_test)
    temp_lst2.append(y_hat)
    temp_lst2.append(X_test)
    temp_lst.append(temp_lst2)
```

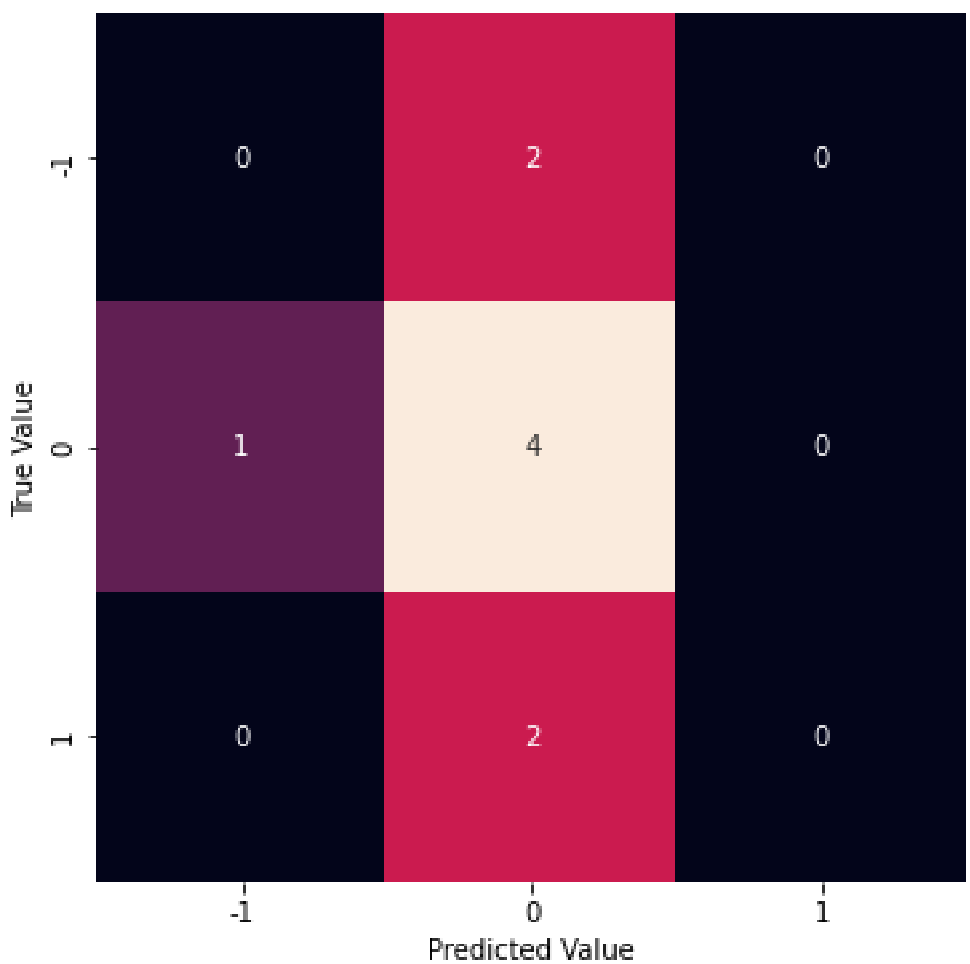
[52]:
```
temp_lst_ytest, temp_lst_yhat, temp_lst_xtest = [], [], []
for row in temp_lst:
    for i in row[4]:
        temp_lst_ytest.append(i)
    for j in row[5]:
        temp_lst_yhat.append(j)
    for k in row[6]:
        temp_lst_xtest.append(k)

cnf_ytest = np.array(temp_lst_ytest)
cnf_yhat = np.array(temp_lst_yhat)
cnf_xtest = np.array(temp_lst_xtest)
print(metrics.classification_report(cnf_ytest, cnf_yhat))
```

```
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         2
           0       0.50      0.80      0.62         5
```

```
           1       0.00      0.00      0.00         2

    accuracy                           0.44         9
   macro avg       0.17      0.27      0.21         9
weighted avg       0.28      0.44      0.34         9
```

[53]:

```python
cnf_matrix = confusion_matrix(cnf_ytest, cnf_yhat)
plt.figure(figsize=(8,6))
sns.heatmap(cnf_matrix, square=True, annot=True, cbar=False)
plt.xticks([.5, 1.5, 2.5], [-1, 0, 1])
plt.yticks([.5, 1.5, 2.5], [-1, 0, 1])
plt.xlabel('Predicted Value')
plt.ylabel('True Value')
plt.show()
```



1.1.3 K-Nearest Neighbor

Advertising Posts

[54]:
```
from sklearn.neighbors import KNeighborsClassifier
from matplotlib.colors import ListedColormap
```

[55]:
```
weights_lst = ['uniform', 'distance']
```

[56]:
```
temp_lst = []
for i in range(2, 6):
    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 n_neighbor in range(1, 10):
            for weight_type in weights_lst:
                clf_knn = KNeighborsClassifier(n_neighbors=n_neighbor,␣
↪weights=weight_type)
                clf_knn.fit(X_train, y_train)
                y_hat = clf_knn.predict(X_test)
                temp_lst2 = []
                temp_lst2.append(i)
                temp_lst2.append(n_neighbor)
                temp_lst2.append(weight_type)
                temp_lst2.append(metrics.accuracy_score(y_train, clf_knn.
↪predict(X_train)))
                temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
                temp_lst2.append(metrics.f1_score(y_test, y_hat))
                temp_lst2.append(metrics.jaccard_score(y_test, y_hat))
                temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst,
                       columns=['k', 'Number of Neighbors', 'Weight Type',␣
↪'Train-set Accuracy', 'Test-set Accuracy', 'F1 Score', 'Jaccard Score'])
temp_lst = []
for k in range(2, 6):
    for n_neighbor in range(1, 10):
        for weight_type in weights_lst:
            temp_lst2 = []
            temp_lst2.append(k)
            temp_lst2.append(n_neighbor)
            temp_lst2.append(weight_type)
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['Number of Neighbors'] == n_neighbor) & (temp_df['Weight Type'] ==␣
↪weight_type)]['Train-set Accuracy']), decimals=4))
```

```
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['Number of Neighbors'] == n_neighbor) & (temp_df['Weight Type'] ==␣
↪weight_type)]['Test-set Accuracy']), decimals=4))
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['Number of Neighbors'] == n_neighbor) & (temp_df['Weight Type'] ==␣
↪weight_type)]['F1 Score']), decimals=4))
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['Number of Neighbors'] == n_neighbor) & (temp_df['Weight Type'] ==␣
↪weight_type)]['Jaccard Score']), decimals=4))
            temp_lst.append(temp_lst2)

clf_knn_eval_df = pd.DataFrame(temp_lst,
                              columns=['k', 'Number of Neighbors', 'Weight␣
↪Type', 'Train-set Accuracy', 'Test-set Accuracy', 'F1 Score', 'Jaccard␣
↪Score'])
clf_knn_eval_df
```

[56]:

	k	Number of Neighbors	Weight Type	Train-set Accuracy	Test-set Accuracy \
0	2	1	uniform	1.00	0.55
1	2	1	distance	1.00	0.55
2	2	2	uniform	0.85	0.63
3	2	2	distance	1.00	0.55
4	2	3	uniform	0.85	0.59
5	2	3	distance	1.00	0.63
6	2	4	uniform	0.85	0.59
7	2	4	distance	1.00	0.59
8	2	5	uniform	0.77	0.62
9	2	5	distance	1.00	0.55
10	2	6	uniform	0.74	0.45
11	2	6	distance	1.00	0.55
12	2	7	uniform	0.66	0.55
13	2	7	distance	1.00	0.59
14	2	8	uniform	0.74	0.41
15	2	8	distance	1.00	0.63
16	2	9	uniform	0.67	0.41
17	2	9	distance	1.00	0.63
18	3	1	uniform	1.00	0.59
19	3	1	distance	1.00	0.59
20	3	2	uniform	0.83	0.63
21	3	2	distance	1.00	0.59
22	3	3	uniform	0.81	0.70
23	3	3	distance	1.00	0.74
24	3	4	uniform	0.78	0.59
25	3	4	distance	1.00	0.63
26	3	5	uniform	0.69	0.63
27	3	5	distance	1.00	0.63
28	3	6	uniform	0.69	0.52

29	3	6	distance	1.00	0.70
30	3	7	uniform	0.67	0.74
31	3	7	distance	1.00	0.74
32	3	8	uniform	0.74	0.70
33	3	8	distance	1.00	0.74
34	3	9	uniform	0.65	0.67
35	3	9	distance	1.00	0.74
36	4	1	uniform	1.00	0.62
37	4	1	distance	1.00	0.62
38	4	2	uniform	0.85	0.65
39	4	2	distance	1.00	0.62
40	4	3	uniform	0.83	0.61
41	4	3	distance	1.00	0.73
42	4	4	uniform	0.84	0.54
43	4	4	distance	1.00	0.58
44	4	5	uniform	0.73	0.58
45	4	5	distance	1.00	0.54
46	4	6	uniform	0.73	0.61
47	4	6	distance	1.00	0.61
48	4	7	uniform	0.68	0.68
49	4	7	distance	1.00	0.65
50	4	8	uniform	0.75	0.65
51	4	8	distance	1.00	0.65
52	4	9	uniform	0.69	0.58
53	4	9	distance	1.00	0.65
54	5	1	uniform	1.00	0.73
55	5	1	distance	1.00	0.73
56	5	2	uniform	0.84	0.67
57	5	2	distance	1.00	0.73
58	5	3	uniform	0.84	0.52
59	5	3	distance	1.00	0.66
60	5	4	uniform	0.84	0.59
61	5	4	distance	1.00	0.55
62	5	5	uniform	0.77	0.52
63	5	5	distance	1.00	0.58
64	5	6	uniform	0.74	0.59
65	5	6	distance	1.00	0.58
66	5	7	uniform	0.69	0.62
67	5	7	distance	1.00	0.61
68	5	8	uniform	0.74	0.65
69	5	8	distance	1.00	0.58
70	5	9	uniform	0.70	0.58
71	5	9	distance	1.00	0.61

	F1 Score	Jaccard Score
0	0.65	0.48
1	0.65	0.48

2	0.69	0.53
3	0.65	0.48
4	0.67	0.51
5	0.69	0.53
6	0.67	0.51
7	0.67	0.50
8	0.73	0.58
9	0.65	0.48
10	0.42	0.29
11	0.65	0.48
12	0.62	0.45
13	0.67	0.51
14	0.32	0.23
15	0.69	0.53
16	0.41	0.28
17	0.69	0.53
18	0.65	0.53
19	0.65	0.53
20	0.68	0.56
21	0.65	0.53
22	0.78	0.68
23	0.80	0.72
24	0.65	0.51
25	0.70	0.58
26	0.72	0.59
27	0.71	0.57
28	0.50	0.44
29	0.78	0.67
30	0.80	0.72
31	0.80	0.72
32	0.78	0.67
33	0.80	0.72
34	0.76	0.65
35	0.80	0.72
36	0.60	0.47
37	0.60	0.47
38	0.62	0.49
39	0.60	0.47
40	0.68	0.57
41	0.74	0.64
42	0.56	0.42
43	0.63	0.50
44	0.66	0.53
45	0.60	0.46
46	0.69	0.58
47	0.68	0.56
48	0.74	0.65

```
49       0.70           0.60
50       0.71           0.60
51       0.70           0.60
52       0.67           0.54
53       0.70           0.60
54       0.72           0.61
55       0.72           0.61
56       0.63           0.50
57       0.72           0.61
58       0.61           0.48
59       0.67           0.55
60       0.66           0.53
61       0.61           0.49
62       0.61           0.48
63       0.64           0.53
64       0.66           0.53
65       0.64           0.53
66       0.70           0.60
67       0.67           0.57
68       0.72           0.61
69       0.64           0.53
70       0.67           0.56
71       0.67           0.57
```

[57]:

```
clf_knn_eval_df.nlargest(3, 'Test-set Accuracy')
```

[57]:

```
    k  Number of Neighbors Weight Type  Train-set Accuracy  Test-set Accuracy  \
23  3                    3    distance                1.00               0.74
30  3                    7     uniform                0.67               0.74
31  3                    7    distance                1.00               0.74

    F1 Score  Jaccard Score
23      0.80           0.72
30      0.80           0.72
31      0.80           0.72
```

[58]:

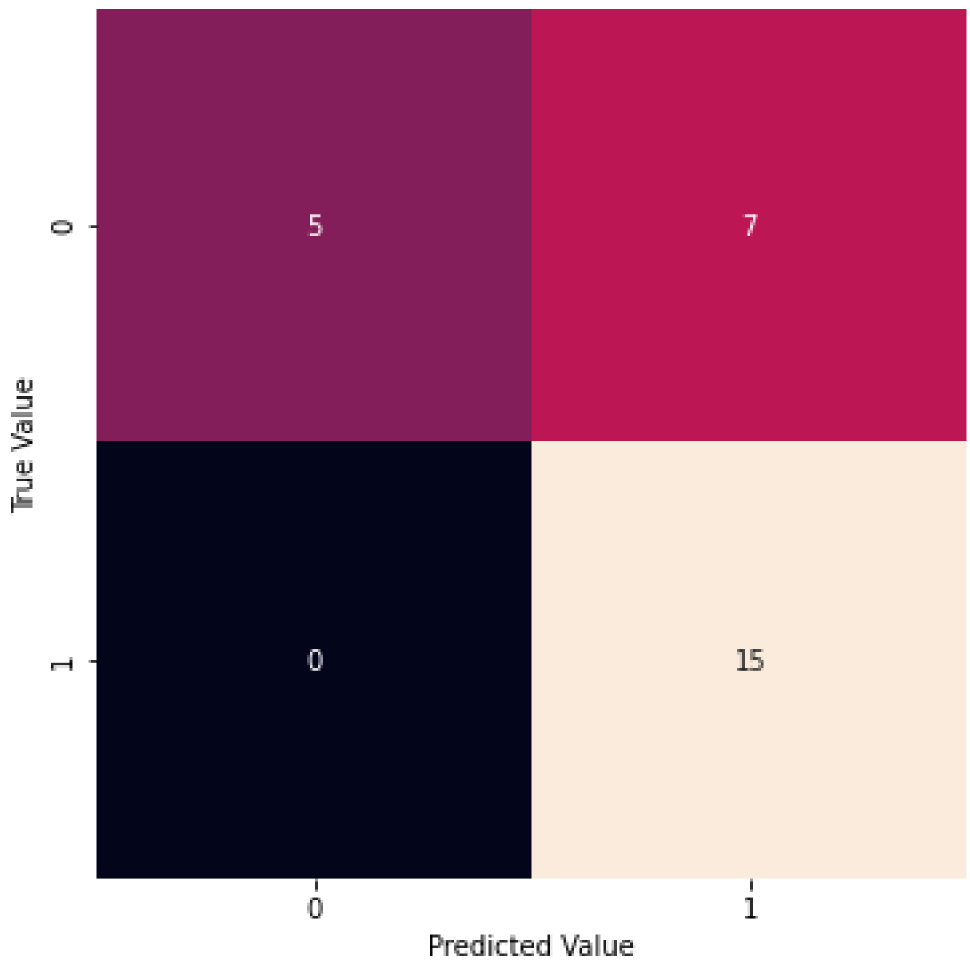
```
kf = KFold(n_splits = 3)
temp_lst = []
clf_knn = KNeighborsClassifier(n_neighbors=3, weights='distance')
for train_index, test_index in kf.split(ad_post_x):
    X_train, X_test = ad_post_x[train_index], ad_post_x[test_index]
    y_train, y_test = ad_post_y[train_index], ad_post_y[test_index]
    clf_knn.fit(X_train, y_train)
    y_hat = clf_knn.predict(X_test)
    temp_lst2 = []
    temp_lst2.append(metrics.accuracy_score(y_train, clf_knn.predict(X_train)))
    temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
```

```python
    temp_lst2.append(metrics.f1_score(y_test, y_hat))
    temp_lst2.append(metrics.jaccard_score(y_test, y_hat))
    temp_lst2.append(y_test)
    temp_lst2.append(y_hat)
    temp_lst2.append(X_test)
    temp_lst.append(temp_lst2)
```

[59]:
```python
temp_lst_ytest, temp_lst_yhat, temp_lst_xtest = [], [], []
for row in temp_lst:
    for i in row[4]:
        temp_lst_ytest.append(i)
    for j in row[5]:
        temp_lst_yhat.append(j)
    for k in row[6]:
        temp_lst_xtest.append(k)

cnf_ytest = np.array(temp_lst_ytest)
cnf_yhat = np.array(temp_lst_yhat)
cnf_xtest = np.array(temp_lst_xtest)
print(metrics.classification_report(cnf_ytest, cnf_yhat))
```

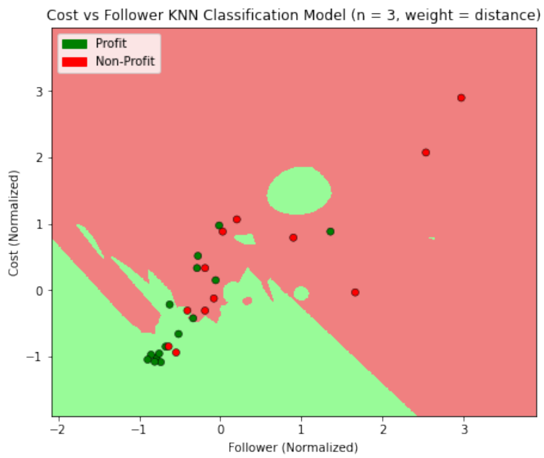
```
              precision    recall  f1-score   support

           0       1.00      0.42      0.59        12
           1       0.68      1.00      0.81        15

    accuracy                           0.74        27
   macro avg       0.84      0.71      0.70        27
weighted avg       0.82      0.74      0.71        27
```

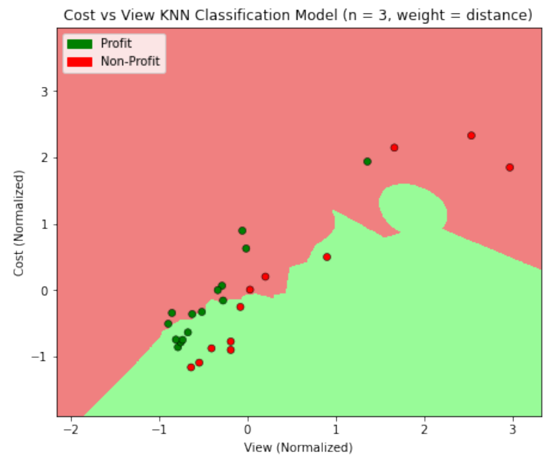
[60]:
```python
cnf_matrix = confusion_matrix(cnf_ytest, cnf_yhat)
plt.figure(figsize=(8,6))
sns.heatmap(cnf_matrix, square=True, annot=True, cbar=False)
# plt.xticks([.5, 1.5, 2.5], [-1, 0, 1])
# plt.yticks([.5, 1.5, 2.5], [-1, 0, 1])
plt.xlabel('Predicted Value')
plt.ylabel('True Value')
plt.show()
```



[61]:
```
fig = plt.figure(figsize = (16, 6))
ax1 = fig.add_subplot(1,2,1)
ax2 = fig.add_subplot(1,2,2)

X,y = ad_post_x[:,:3], ad_post_y
h = .01
cmap_light = ListedColormap(['lightcoral', 'palegreen'])
cmap_bold = ['r', 'g']
X_follower = X[:,(0,2)]
X_view = X[:,(1,2)]

xx_follower, xx_view = None, None
yy_follower, yy_view = None, None
Z_follower, Z_view = None, None
g1, g2 = None, None
```

```
X_lst = [X_follower, X_view]
xx_lst = [xx_follower, xx_view]
yy_lst = [yy_follower, yy_view]
Z_lst = [Z_follower, Z_view]
ax_lst = [ax1, ax2]
g_lst = [g1, g2]
x_label_lst = ['Follower', 'View']
labels=['Non-Profit','Profit']
red_patch = patches.Patch(color='r', label='Non-Profit')
green_patch = patches.Patch(color='g', label='Profit')


def plot_calc(x, y = y):
    '''
    This function is for calculating the area to plot with colors according to␣
 ↪the input
        input -> x and y.
        return -> xx, yy, Z which are needed to drawing the contour and plot.
    '''
    clf_knn.fit(x, y)
    x_min, x_max = x[:, 0].min() - 1, x[:, 0].max() + 1
    y_min, y_max = x[:, 1].min() - 1, x[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = clf_knn.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    return xx, yy, Z

for ax_, xx_, yy_, Z_, X_, x_label_, g_ in zip(ax_lst, xx_lst, yy_lst, Z_lst,␣
 ↪X_lst, x_label_lst, g_lst):
    xx_, yy_, Z_ = plot_calc(X_, y)
    ax_.contourf(xx_, yy_, Z_, cmap=cmap_light)
    g_ = sns.scatterplot(x=X_[:, 1], y=X_[:, 0], hue=ad_post['benefit'],␣
 ↪palette=cmap_bold, alpha=1.0, edgecolor='black', ax=ax_)
    ax_.set_title(f'Cost vs {x_label_} KNN Classification Model (n = 3, weight␣
 ↪= distance)')
    ax_.set_xlim(xx_.min(), xx_.max())
    ax_.set_ylim(yy_.min(), yy_.max())
    ax_.set_xlabel(f'{x_label_} (Normalized)')
    ax_.set_ylabel('Cost (Normalized)')
    ax_.legend(handles=[green_patch, red_patch],loc = 'upper left', fontsize =␣
 ↪10);

plt.show()
```





Advertising Story

[62]:
```
temp_lst = []
for i in range(2, 6):
    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 n_neighbor in range(1, 10):
            for weight_type in weights_lst:
                clf_knn = KNeighborsClassifier(n_neighbors=n_neighbor,␣
 ↪weights=weight_type)
                clf_knn.fit(X_train, y_train)
                y_hat = clf_knn.predict(X_test)
                temp_lst2 = []
                temp_lst2.append(i)
                temp_lst2.append(n_neighbor)
                temp_lst2.append(weight_type)
                temp_lst2.append(metrics.accuracy_score(y_train, clf_knn.
 ↪predict(X_train)))
                temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
                temp_lst2.append(metrics.f1_score(y_test, y_hat))
                temp_lst2.append(metrics.jaccard_score(y_test, y_hat))
                temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst,
                       columns=['k', 'Number of Neighbors', 'Weight Type',␣
 ↪'Train-set Accuracy', 'Test-set Accuracy', 'F1 Score', 'Jaccard Score'])
temp_lst = []
for k in range(2, 6):
    for n_neighbor in range(1, 10):
```

```python
        for weight_type in weights_lst:
            temp_lst2 = []
            temp_lst2.append(k)
            temp_lst2.append(n_neighbor)
            temp_lst2.append(weight_type)
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['Number of Neighbors'] == n_neighbor) & (temp_df['Weight Type'] ==␣
↪weight_type)]['Train-set Accuracy']), decimals=4))
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['Number of Neighbors'] == n_neighbor) & (temp_df['Weight Type'] ==␣
↪weight_type)]['Test-set Accuracy']), decimals=4))
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['Number of Neighbors'] == n_neighbor) & (temp_df['Weight Type'] ==␣
↪weight_type)]['F1 Score']), decimals=4))
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['Number of Neighbors'] == n_neighbor) & (temp_df['Weight Type'] ==␣
↪weight_type)]['Jaccard Score']), decimals=4))
            temp_lst.append(temp_lst2)

clf_knn_eval_df = pd.DataFrame(temp_lst,
                              columns=['k', 'Number of Neighbors', 'Weight␣
↪Type', 'Train-set Accuracy', 'Test-set Accuracy', 'F1 Score', 'Jaccard␣
↪Score'])
clf_knn_eval_df
```

[62]:

	k	Number of Neighbors	Weight Type	Train-set Accuracy	Test-set Accuracy \
0	2	1	uniform	1.00	0.78
1	2	1	distance	1.00	0.78
2	2	2	uniform	0.89	0.78
3	2	2	distance	1.00	0.78
4	2	3	uniform	0.74	0.62
5	2	3	distance	1.00	0.74
6	2	4	uniform	0.70	0.66
7	2	4	distance	1.00	0.70
8	2	5	uniform	0.74	0.70
9	2	5	distance	1.00	0.74
10	2	6	uniform	0.74	0.66
11	2	6	distance	1.00	0.70
12	2	7	uniform	0.70	0.59
13	2	7	distance	1.00	0.70
14	2	8	uniform	0.70	0.52
15	2	8	distance	1.00	0.70
16	2	9	uniform	0.70	0.59
17	2	9	distance	1.00	0.66
18	3	1	uniform	1.00	0.74
19	3	1	distance	1.00	0.74
20	3	2	uniform	0.89	0.74

21	3	2	distance	1.00	0.74
22	3	3	uniform	0.81	0.67
23	3	3	distance	1.00	0.78
24	3	4	uniform	0.83	0.74
25	3	4	distance	1.00	0.78
26	3	5	uniform	0.72	0.70
27	3	5	distance	1.00	0.78
28	3	6	uniform	0.76	0.70
29	3	6	distance	1.00	0.81
30	3	7	uniform	0.63	0.59
31	3	7	distance	1.00	0.78
32	3	8	uniform	0.65	0.59
33	3	8	distance	1.00	0.81
34	3	9	uniform	0.61	0.59
35	3	9	distance	1.00	0.74
36	4	1	uniform	1.00	0.74
37	4	1	distance	1.00	0.74
38	4	2	uniform	0.90	0.74
39	4	2	distance	1.00	0.74
40	4	3	uniform	0.83	0.66
41	4	3	distance	1.00	0.78
42	4	4	uniform	0.83	0.70
43	4	4	distance	1.00	0.74
44	4	5	uniform	0.73	0.70
45	4	5	distance	1.00	0.71
46	4	6	uniform	0.75	0.66
47	4	6	distance	1.00	0.74
48	4	7	uniform	0.65	0.58
49	4	7	distance	1.00	0.70
50	4	8	uniform	0.72	0.58
51	4	8	distance	1.00	0.77
52	4	9	uniform	0.64	0.58
53	4	9	distance	1.00	0.67
54	5	1	uniform	1.00	0.75
55	5	1	distance	1.00	0.75
56	5	2	uniform	0.90	0.71
57	5	2	distance	1.00	0.75
58	5	3	uniform	0.83	0.67
59	5	3	distance	1.00	0.79
60	5	4	uniform	0.82	0.71
61	5	4	distance	1.00	0.75
62	5	5	uniform	0.74	0.77
63	5	5	distance	1.00	0.79
64	5	6	uniform	0.79	0.77
65	5	6	distance	1.00	0.81
66	5	7	uniform	0.65	0.59
67	5	7	distance	1.00	0.85

68	5	8	uniform	0.69	0.63
69	5	8	distance	1.00	0.81
70	5	9	uniform	0.62	0.63
71	5	9	distance	1.00	0.82

	F1 Score	Jaccard Score
0	0.82	0.69
1	0.82	0.69
2	0.82	0.69
3	0.82	0.69
4	0.75	0.62
5	0.80	0.66
6	0.77	0.64
7	0.77	0.64
8	0.79	0.69
9	0.80	0.66
10	0.77	0.64
11	0.77	0.64
12	0.73	0.59
13	0.77	0.64
14	0.65	0.48
15	0.77	0.64
16	0.72	0.58
17	0.75	0.61
18	0.78	0.64
19	0.78	0.64
20	0.78	0.64
21	0.78	0.64
22	0.76	0.61
23	0.82	0.71
24	0.80	0.67
25	0.82	0.71
26	0.79	0.68
27	0.82	0.71
28	0.79	0.68
29	0.86	0.75
30	0.73	0.59
31	0.83	0.71
32	0.73	0.59
33	0.86	0.75
34	0.73	0.59
35	0.80	0.67
36	0.74	0.61
37	0.74	0.61
38	0.74	0.61
39	0.74	0.61
40	0.74	0.60

```
41       0.82           0.70
42       0.76           0.63
43       0.79           0.66
44       0.76           0.64
45       0.78           0.64
46       0.74           0.61
47       0.79           0.67
48       0.71           0.58
49       0.77           0.63
50       0.71           0.57
51       0.81           0.69
52       0.71           0.58
53       0.74           0.60
54       0.76           0.65
55       0.76           0.65
56       0.72           0.58
57       0.76           0.65
58       0.75           0.61
59       0.82           0.72
60       0.77           0.63
61       0.80           0.67
62       0.81           0.70
63       0.82           0.72
64       0.81           0.70
65       0.84           0.73
66       0.71           0.59
67       0.87           0.78
68       0.74           0.62
69       0.84           0.73
70       0.74           0.62
71       0.85           0.75
```

[63]:
```
clf_knn_eval_df.nlargest(3, 'Test-set Accuracy')
```

[63]:
```
    k  Number of Neighbors Weight Type  Train-set Accuracy  Test-set Accuracy  \
67  5                    7    distance                1.00               0.85
71  5                    9    distance                1.00               0.82
29  3                    6    distance                1.00               0.81

    F1 Score  Jaccard Score
67      0.87           0.78
71      0.85           0.75
29      0.86           0.75
```

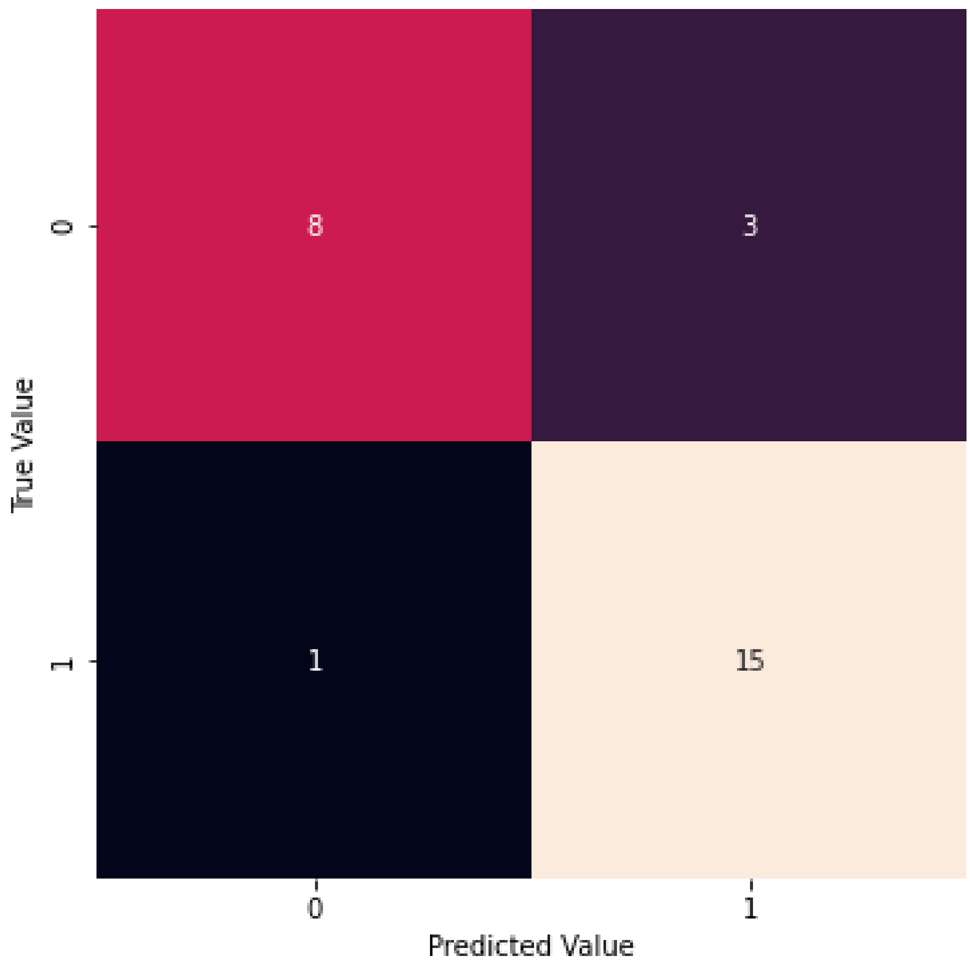
[64]:
```
kf = KFold(n_splits = 5)
temp_lst = []
clf_knn = KNeighborsClassifier(n_neighbors=7, weights='distance')
```

```
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]
    clf_knn.fit(X_train, y_train)
    y_hat = clf_knn.predict(X_test)
    temp_lst2 = []
    temp_lst2.append(metrics.accuracy_score(y_train, clf_knn.predict(X_train)))
    temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
    temp_lst2.append(metrics.f1_score(y_test, y_hat))
    temp_lst2.append(metrics.jaccard_score(y_test, y_hat))
    temp_lst2.append(y_test)
    temp_lst2.append(y_hat)
    temp_lst2.append(X_test)
    temp_lst.append(temp_lst2)
```

[65]:
```
temp_lst_ytest, temp_lst_yhat, temp_lst_xtest = [], [], []
for row in temp_lst:
    for i in row[4]:
        temp_lst_ytest.append(i)
    for j in row[5]:
        temp_lst_yhat.append(j)
    for k in row[6]:
        temp_lst_xtest.append(k)

cnf_ytest = np.array(temp_lst_ytest)
cnf_yhat = np.array(temp_lst_yhat)
cnf_xtest = np.array(temp_lst_xtest)
print(metrics.classification_report(cnf_ytest, cnf_yhat))
```

```
              precision    recall  f1-score   support

           0       0.89      0.73      0.80        11
           1       0.83      0.94      0.88        16

    accuracy                           0.85        27
   macro avg       0.86      0.83      0.84        27
weighted avg       0.86      0.85      0.85        27
```

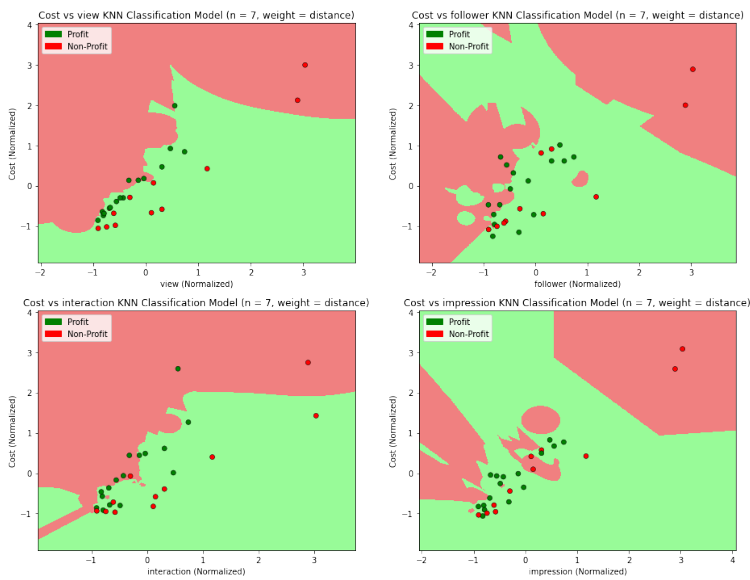
[66]:
```
cnf_matrix = confusion_matrix(cnf_ytest, cnf_yhat)
plt.figure(figsize=(8,6))
sns.heatmap(cnf_matrix, square=True, annot=True, cbar=False)
# plt.xticks([.5, 1.5, 2.5], [-1, 0, 1])
# plt.yticks([.5, 1.5, 2.5], [-1, 0, 1])
plt.xlabel('Predicted Value')
plt.ylabel('True Value')
plt.show()
```



[67]:
```
fig = plt.figure(figsize = (24, 12))
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)


X,y = ad_story_x[:,:6], ad_story_y
h = .01
cmap_light = ListedColormap(['lightcoral', 'palegreen'])
cmap_bold = ['r', 'g']
X_view = X[:,(0,5)]
X_follower = X[:,(1,5)]
X_action = X[:,(2,5)]
X_interaction = X[:,(3,5)]
```

```python
X_impression = X[:,(4,5)]


xx_view, xx_follower, xx_action, xx_interaction, xx_impression = None, None,␣
↪None, None, None
yy_view, yy_follower, yy_action, yy_interaction, yy_impression = None, None,␣
↪None, None, None
Z_view, Z_follower, Z_action, Z_interaction, Z_impression = None, None, None,␣
↪None, None
g1, g2, g3, g4, g5 = None, None, None, None, None

X_lst = [X_view, X_follower, X_action, X_interaction, X_impression]
xx_lst = [xx_view, xx_follower, xx_action, xx_interaction, xx_impression]
yy_lst = [yy_view, yy_follower, yy_action, yy_interaction, yy_impression]
Z_lst = [Z_view, Z_follower, Z_action, Z_interaction, Z_impression]
ax_lst = [ax1, ax2, ax3, ax4, ax5]
g_lst = [g1, g2, g3, g4, g5]
x_label_lst = ['view', 'follower', 'action', 'interaction', 'impression']
labels=['Non-Profit','Profit']
red_patch = patches.Patch(color='r', label='Non-Profit')
green_patch = patches.Patch(color='g', label='Profit')


def plot_calc(x, y = y):
    '''
    This function is for calculating the area to plot with colors according to␣
↪the input
        input -> x and y.
        return -> xx, yy, Z which are needed to drawing the contour and plot.
    '''
    clf_knn.fit(x, y)
    x_min, x_max = x[:, 0].min() - 1, x[:, 0].max() + 1
    y_min, y_max = x[:, 1].min() - 1, x[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = clf_knn.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    return xx, yy, Z

for ax_, xx_, yy_, Z_, X_, x_label_, g_ in zip(ax_lst, xx_lst, yy_lst, Z_lst,␣
↪X_lst, x_label_lst, g_lst):
    xx_, yy_, Z_ = plot_calc(X_, y)
    ax_.contourf(xx_, yy_, Z_, cmap=cmap_light)
    g_ = sns.scatterplot(x=X_[:, 1], y=X_[:, 0], hue=ad_story['benefit'],␣
↪palette=cmap_bold, alpha=1.0, edgecolor='black', ax=ax_)
    ax_.set_title(f'Cost vs {x_label_} KNN Classification Model (n = 7, weight␣
↪= distance)')
```

```
    ax_.set_xlim(xx_.min(), xx_.max())
    ax_.set_ylim(yy_.min(), yy_.max())
    ax_.set_xlabel(f'{x_label_} (Normalized)')
    ax_.set_ylabel('Cost (Normalized)')
    ax_.legend(handles=[green_patch, red_patch],loc = 'upper left', fontsize = 10);

plt.show()
```



Influencer

[68]:
```
temp_lst = []
for i in range(2, 6):
    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 n_neighbor in range(1, 10):
            for weight_type in weights_lst:
                clf_knn = KNeighborsClassifier(n_neighbors=n_neighbor, weights=weight_type)
                clf_knn.fit(X_train, y_train)
                y_hat = clf_knn.predict(X_test)
                temp_lst2 = []
                temp_lst2.append(i)
                temp_lst2.append(n_neighbor)
                temp_lst2.append(weight_type)
```

```
                temp_lst2.append(metrics.accuracy_score(y_train, clf_knn.
↪predict(X_train)))
                temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
                temp_lst2.append(metrics.f1_score(y_test, y_hat,␣
↪average='micro'))
                temp_lst2.append(metrics.jaccard_score(y_test, y_hat,␣
↪average='micro'))
                temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst,
                       columns=['k', 'Number of Neighbors', 'Weight Type',␣
↪'Train-set Accuracy', 'Test-set Accuracy', 'F1 Score', 'Jaccard Score'])
temp_lst = []
for k in range(2, 6):
    for n_neighbor in range(1, 10):
        for weight_type in weights_lst:
            temp_lst2 = []
            temp_lst2.append(k)
            temp_lst2.append(n_neighbor)
            temp_lst2.append(weight_type)
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['Number of Neighbors'] == n_neighbor) & (temp_df['Weight Type'] ==␣
↪weight_type)]['Train-set Accuracy']), decimals=4))
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['Number of Neighbors'] == n_neighbor) & (temp_df['Weight Type'] ==␣
↪weight_type)]['Test-set Accuracy']), decimals=4))
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['Number of Neighbors'] == n_neighbor) & (temp_df['Weight Type'] ==␣
↪weight_type)]['F1 Score']), decimals=4))
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['Number of Neighbors'] == n_neighbor) & (temp_df['Weight Type'] ==␣
↪weight_type)]['Jaccard Score']), decimals=4))
            temp_lst.append(temp_lst2)

clf_knn_eval_df = pd.DataFrame(temp_lst,
                               columns=['k', 'Number of Neighbors', 'Weight␣
↪Type', 'Train-set Accuracy', 'Test-set Accuracy', 'F1 Score', 'Jaccard␣
↪Score'])
clf_knn_eval_df
```

[68]:

```
   k  Number of Neighbors Weight Type  Train-set Accuracy  Test-set Accuracy  \
0  2                    1     uniform                1.00               0.61
1  2                    1    distance                1.00               0.61
2  2                    2     uniform                0.96               0.67
3  2                    2    distance                1.00               0.61
4  2                    3     uniform                0.97               0.60
```

5	2	3	distance	1.00	0.58
6	2	4	uniform	0.89	0.61
7	2	4	distance	1.00	0.60
8	2	5	uniform	0.84	0.65
9	2	5	distance	1.00	0.56
10	2	6	uniform	0.77	0.63
11	2	6	distance	1.00	0.65
12	2	7	uniform	0.78	0.62
13	2	7	distance	1.00	0.67
14	2	8	uniform	0.75	0.53
15	2	8	distance	1.00	0.70
16	2	9	uniform	0.70	0.48
17	2	9	distance	1.00	0.59
18	3	1	uniform	1.00	0.48
19	3	1	distance	1.00	0.48
20	3	2	uniform	0.96	0.50
21	3	2	distance	1.00	0.48
22	3	3	uniform	0.97	0.53
23	3	3	distance	1.00	0.51
24	3	4	uniform	0.88	0.45
25	3	4	distance	1.00	0.51
26	3	5	uniform	0.87	0.51
27	3	5	distance	1.00	0.51
28	3	6	uniform	0.83	0.47
29	3	6	distance	1.00	0.53
30	3	7	uniform	0.80	0.45
31	3	7	distance	1.00	0.47
32	3	8	uniform	0.81	0.39
33	3	8	distance	1.00	0.43
34	3	9	uniform	0.78	0.39
35	3	9	distance	1.00	0.41
36	4	1	uniform	1.00	0.54
37	4	1	distance	1.00	0.54
38	4	2	uniform	0.95	0.52
39	4	2	distance	1.00	0.54
40	4	3	uniform	0.96	0.52
41	4	3	distance	1.00	0.57
42	4	4	uniform	0.88	0.44
43	4	4	distance	1.00	0.57
44	4	5	uniform	0.87	0.51
45	4	5	distance	1.00	0.57
46	4	6	uniform	0.82	0.45
47	4	6	distance	1.00	0.60
48	4	7	uniform	0.80	0.45
49	4	7	distance	1.00	0.57
50	4	8	uniform	0.79	0.37
51	4	8	distance	1.00	0.50

52	4	9	uniform	0.78	0.33
53	4	9	distance	1.00	0.44
54	5	1	uniform	1.00	0.52
55	5	1	distance	1.00	0.52
56	5	2	uniform	0.95	0.52
57	5	2	distance	1.00	0.52
58	5	3	uniform	0.96	0.53
59	5	3	distance	1.00	0.53
60	5	4	uniform	0.87	0.45
61	5	4	distance	1.00	0.53
62	5	5	uniform	0.86	0.50
63	5	5	distance	1.00	0.53
64	5	6	uniform	0.84	0.44
65	5	6	distance	1.00	0.54
66	5	7	uniform	0.79	0.44
67	5	7	distance	1.00	0.47
68	5	8	uniform	0.80	0.40
69	5	8	distance	1.00	0.46
70	5	9	uniform	0.79	0.32
71	5	9	distance	1.00	0.39

	F1 Score	Jaccard Score
0	0.61	0.44
1	0.61	0.44
2	0.67	0.50
3	0.61	0.44
4	0.60	0.43
5	0.58	0.41
6	0.61	0.44
7	0.60	0.43
8	0.65	0.48
9	0.56	0.39
10	0.63	0.46
11	0.65	0.48
12	0.62	0.45
13	0.67	0.50
14	0.53	0.36
15	0.70	0.53
16	0.48	0.32
17	0.59	0.42
18	0.48	0.33
19	0.48	0.33
20	0.50	0.34
21	0.48	0.33
22	0.53	0.37
23	0.51	0.35
24	0.45	0.29

25	0.51	0.35
26	0.51	0.34
27	0.51	0.34
28	0.47	0.32
29	0.53	0.36
30	0.45	0.30
31	0.47	0.31
32	0.39	0.25
33	0.43	0.28
34	0.39	0.25
35	0.41	0.26
36	0.54	0.39
37	0.54	0.39
38	0.52	0.38
39	0.54	0.39
40	0.52	0.36
41	0.57	0.41
42	0.44	0.29
43	0.57	0.41
44	0.51	0.35
45	0.57	0.41
46	0.45	0.30
47	0.60	0.43
48	0.45	0.30
49	0.57	0.40
50	0.37	0.24
51	0.50	0.34
52	0.33	0.21
53	0.44	0.29
54	0.52	0.39
55	0.52	0.39
56	0.52	0.38
57	0.52	0.39
58	0.53	0.38
59	0.53	0.38
60	0.45	0.31
61	0.53	0.38
62	0.50	0.35
63	0.53	0.38
64	0.44	0.31
65	0.54	0.38
66	0.44	0.30
67	0.47	0.32
68	0.40	0.28
69	0.46	0.31
70	0.32	0.22
71	0.39	0.25

[69]:
```
clf_knn_eval_df.nlargest(3, 'Test-set Accuracy')
```

[69]:
```
    k  Number of Neighbors Weight Type  Train-set Accuracy  Test-set Accuracy  \
15  2                    8    distance                1.00               0.70
2   2                    2     uniform                0.96               0.67
13  2                    7    distance                1.00               0.67

    F1 Score  Jaccard Score
15      0.70           0.53
2       0.67           0.50
13      0.67           0.50
```

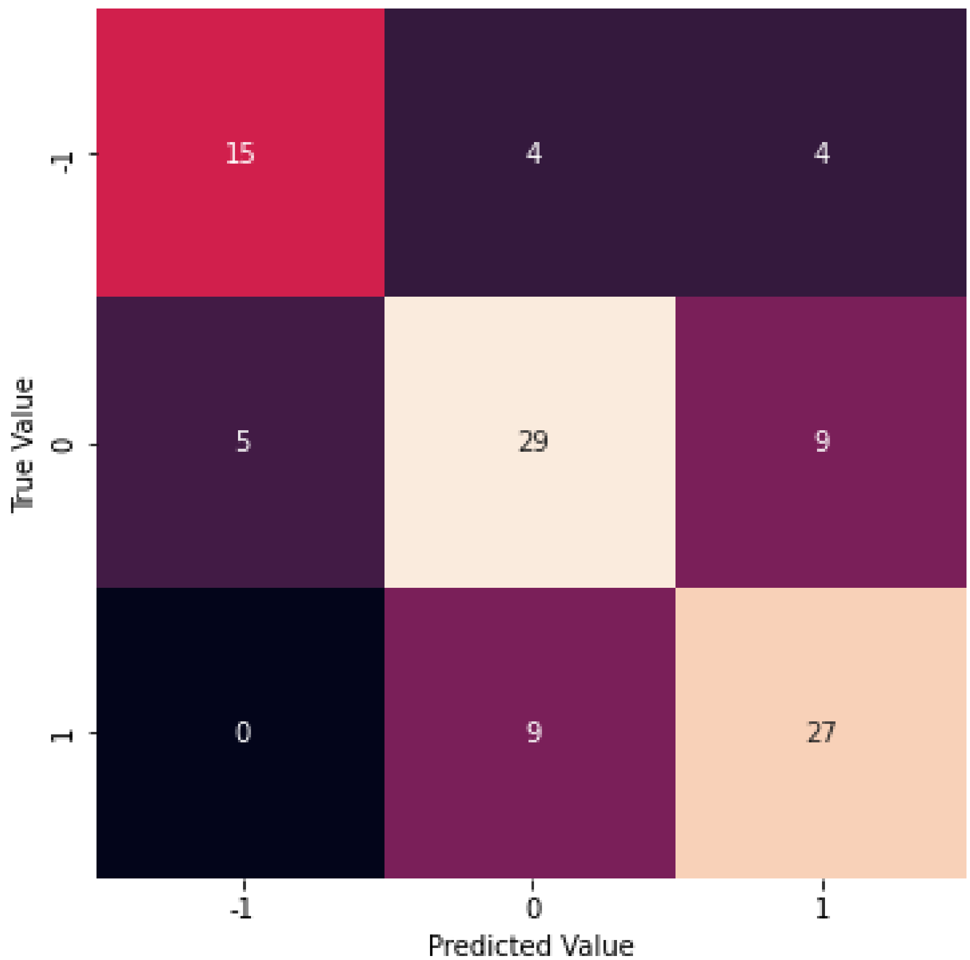
[70]:
```
kf = KFold(n_splits = 2)
temp_lst = []
clf_knn = KNeighborsClassifier(n_neighbors=8, weights='distance')
for train_index, test_index in kf.split(influencer_x):
    X_train, X_test = influencer_x[train_index], influencer_x[test_index]
    y_train, y_test = influencer_y[train_index], influencer_y[test_index]
    clf_knn.fit(X_train, y_train)
    y_hat = clf_knn.predict(X_test)
    temp_lst2 = []
    temp_lst2.append(metrics.accuracy_score(y_train, clf_knn.predict(X_train)))
    temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
    temp_lst2.append(metrics.f1_score(y_test, y_hat, average='micro'))
    temp_lst2.append(metrics.jaccard_score(y_test, y_hat, average='micro'))
    temp_lst2.append(y_test)
    temp_lst2.append(y_hat)
    temp_lst2.append(X_test)
    temp_lst.append(temp_lst2)
```

[71]:
```
temp_lst_ytest, temp_lst_yhat, temp_lst_xtest = [], [], []
for row in temp_lst:
    for i in row[4]:
        temp_lst_ytest.append(i)
    for j in row[5]:
        temp_lst_yhat.append(j)
    for k in row[6]:
        temp_lst_xtest.append(k)

cnf_ytest = np.array(temp_lst_ytest)
cnf_yhat = np.array(temp_lst_yhat)
cnf_xtest = np.array(temp_lst_xtest)
print(metrics.classification_report(cnf_ytest, cnf_yhat))
```

```
              precision    recall  f1-score   support

          -1       0.75      0.65      0.70        23
           0       0.69      0.67      0.68        43
```

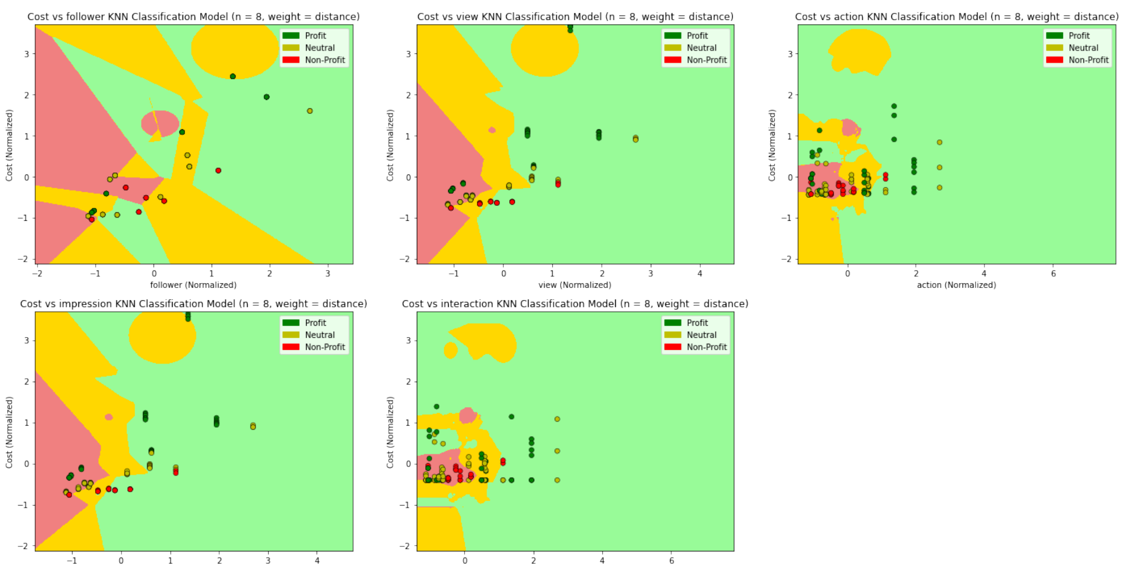
```
           1       0.68      0.75      0.71        36

    accuracy                           0.70       102
   macro avg       0.71      0.69      0.70       102
weighted avg       0.70      0.70      0.70       102
```

[72]:
```
cnf_matrix = confusion_matrix(cnf_ytest, cnf_yhat)
plt.figure(figsize=(8,6))
sns.heatmap(cnf_matrix, square=True, annot=True, cbar=False)
plt.xticks([.5, 1.5, 2.5], [-1, 0, 1])
plt.yticks([.5, 1.5, 2.5], [-1, 0, 1])
plt.xlabel('Predicted Value')
plt.ylabel('True Value')
plt.show()
```



[73]:
```
fig = plt.figure(figsize = (24, 12))
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)


X,y = influencer_x[:,:7], influencer_y
h = .01
cmap_light = ListedColormap(['lightcoral', 'gold', 'palegreen'])
cmap_bold = ['r', 'y', 'g']
X_follower = X[:,(0,6)]
X_view = X[:,(1,6)]
X_action = X[:,(2,6)]
X_impression = X[:,(3,6)]
X_interaction = X[:,(5,6)]


xx_follower, xx_view, xx_action, xx_impression, xx_interaction = None, None,␣
 ↪None, None, None
yy_follower, yy_view, yy_action, yy_impression, yy_interaction = None, None,␣
 ↪None, None, None
Z_follower, Z_view, Z_action, Z_impression, Z_interaction = None, None, None,␣
 ↪None, None
g1, g2, g3, g4, g5 = None, None, None, None, None

X_lst = [X_follower, X_view, X_action, X_impression, X_interaction]
xx_lst = [xx_follower, xx_view, xx_action, xx_impression, xx_interaction]
yy_lst = [yy_follower, yy_view, yy_action, yy_impression, yy_interaction]
Z_lst = [Z_follower, Z_view, Z_action, Z_impression, Z_interaction]
ax_lst = [ax1, ax2, ax3, ax4, ax5]
g_lst = [g1, g2, g3, g4, g5]
x_label_lst = ['follower', 'view', 'action', 'impression', 'interaction']
labels=['Non-Profit','Neutral', 'Profit']
red_patch = patches.Patch(color='r', label='Non-Profit')
yellow_patch = patches.Patch(color='y', label='Neutral')
green_patch = patches.Patch(color='g', label='Profit')


def plot_calc(x, y = y):
    '''
    This function is for calculating the area to plot with colors according to␣
 ↪the input
        input -> x and y.
        return -> xx, yy, Z which are needed to drawing the contour and plot.
    '''
```

```
    clf_knn.fit(x, y)
    x_min, x_max = x[:, 0].min() - 1, x[:, 0].max() + 1
    y_min, y_max = x[:, 1].min() - 1, x[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = clf_knn.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    return xx, yy, Z

for ax_, xx_, yy_, Z_, X_, x_label_, g_ in zip(ax_lst, xx_lst, yy_lst, Z_lst, X_lst, x_label_lst, g_lst):
    xx_, yy_, Z_ = plot_calc(X_, y)
    ax_.contourf(xx_, yy_, Z_, cmap=cmap_light)
    g_ = sns.scatterplot(x=X_[:, 1], y=X_[:, 0], hue=influencer['benefit'], palette=cmap_bold, alpha=1.0, edgecolor='black', ax=ax_)
    ax_.set_title(f'Cost vs {x_label_} KNN Classification Model (n = 8, weight = distance)')
    ax_.set_xlim(xx_.min(), xx_.max())
    ax_.set_ylim(yy_.min(), yy_.max())
    ax_.set_xlabel(f'{x_label_} (Normalized)')
    ax_.set_ylabel('Cost (Normalized)')
    ax_.legend(handles=[green_patch, yellow_patch, red_patch],loc = 'upper right', fontsize = 10);

plt.show()
```



Leaders_post

[74]:

```
temp_lst = []
for i in range(2, 6):
    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 n_neighbor in range(1, 5):
            for weight_type in weights_lst:
                clf_knn = KNeighborsClassifier(n_neighbors=n_neighbor,␣
↪weights=weight_type)
                clf_knn.fit(X_train, y_train)
                y_hat = clf_knn.predict(X_test)
                temp_lst2 = []
                temp_lst2.append(i)
                temp_lst2.append(n_neighbor)
                temp_lst2.append(weight_type)
                temp_lst2.append(metrics.accuracy_score(y_train, clf_knn.
↪predict(X_train)))
                temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
                temp_lst2.append(metrics.f1_score(y_test, y_hat,␣
↪average='micro'))
                temp_lst2.append(metrics.jaccard_score(y_test, y_hat,␣
↪average='micro'))
                temp_lst.append(temp_lst2)

temp_df = pd.DataFrame(temp_lst,
                       columns=['k', 'Number of Neighbors', 'Weight Type',␣
↪'Train-set Accuracy', 'Test-set Accuracy', 'F1 Score', 'Jaccard Score'])
temp_lst = []
for k in range(2, 6):
    for n_neighbor in range(1, 5):
        for weight_type in weights_lst:
            temp_lst2 = []
            temp_lst2.append(k)
            temp_lst2.append(n_neighbor)
            temp_lst2.append(weight_type)
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['Number of Neighbors'] == n_neighbor) & (temp_df['Weight Type'] ==␣
↪weight_type)]['Train-set Accuracy']), decimals=4))
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['Number of Neighbors'] == n_neighbor) & (temp_df['Weight Type'] ==␣
↪weight_type)]['Test-set Accuracy']), decimals=4))
```

```
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['Number of Neighbors'] == n_neighbor) & (temp_df['Weight Type'] ==␣
↪weight_type)]['F1 Score']), decimals=4))
            temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['Number of Neighbors'] == n_neighbor) & (temp_df['Weight Type'] ==␣
↪weight_type)]['Jaccard Score']), decimals=4))
            temp_lst.append(temp_lst2)

clf_knn_eval_df = pd.DataFrame(temp_lst,
                              columns=['k', 'Number of Neighbors', 'Weight␣
↪Type', 'Train-set Accuracy', 'Test-set Accuracy', 'F1 Score', 'Jaccard␣
↪Score'])
clf_knn_eval_df
```

[74]:

	k	Number of Neighbors	Weight Type	Train-set Accuracy	Test-set Accuracy \
0	2	1	uniform	1.00	0.35
1	2	1	distance	1.00	0.35
2	2	2	uniform	0.65	0.35
3	2	2	distance	1.00	0.35
4	2	3	uniform	0.88	0.45
5	2	3	distance	1.00	0.35
6	2	4	uniform	0.78	0.57
7	2	4	distance	1.00	0.45
8	3	1	uniform	1.00	0.56
9	3	1	distance	1.00	0.56
10	3	2	uniform	0.67	0.44
11	3	2	distance	1.00	0.56
12	3	3	uniform	0.72	0.44
13	3	3	distance	1.00	0.44
14	3	4	uniform	0.61	0.22
15	3	4	distance	1.00	0.44
16	4	1	uniform	1.00	0.42
17	4	1	distance	1.00	0.42
18	4	2	uniform	0.70	0.33
19	4	2	distance	1.00	0.42
20	4	3	uniform	0.67	0.46
21	4	3	distance	1.00	0.33
22	4	4	uniform	0.70	0.46
23	4	4	distance	1.00	0.46
24	5	1	uniform	1.00	0.50
25	5	1	distance	1.00	0.50
26	5	2	uniform	0.70	0.50
27	5	2	distance	1.00	0.50
28	5	3	uniform	0.72	0.50
29	5	3	distance	1.00	0.50
30	5	4	uniform	0.67	0.40
31	5	4	distance	1.00	0.50

	F1 Score	Jaccard Score
0	0.35	0.22
1	0.35	0.22
2	0.35	0.22
3	0.35	0.22
4	0.45	0.29
5	0.35	0.22
6	0.57	0.42
7	0.45	0.29
8	0.56	0.40
9	0.56	0.40
10	0.44	0.30
11	0.56	0.40
12	0.44	0.30
13	0.44	0.30
14	0.22	0.13
15	0.44	0.30
16	0.42	0.29
17	0.42	0.29
18	0.33	0.22
19	0.42	0.29
20	0.46	0.30
21	0.33	0.22
22	0.46	0.30
23	0.46	0.30
24	0.50	0.40
25	0.50	0.40
26	0.50	0.40
27	0.50	0.40
28	0.50	0.40
29	0.50	0.40
30	0.40	0.33
31	0.50	0.40

[75]:
```python
clf_knn_eval_df.nlargest(3, 'Test-set Accuracy')
```

[75]:

	k	Number of Neighbors	Weight Type	Train-set Accuracy	Test-set Accuracy	\
6	2	4	uniform	0.78	0.57	
8	3	1	uniform	1.00	0.56	
9	3	1	distance	1.00	0.56	

	F1 Score	Jaccard Score
6	0.57	0.42
8	0.56	0.40
9	0.56	0.40

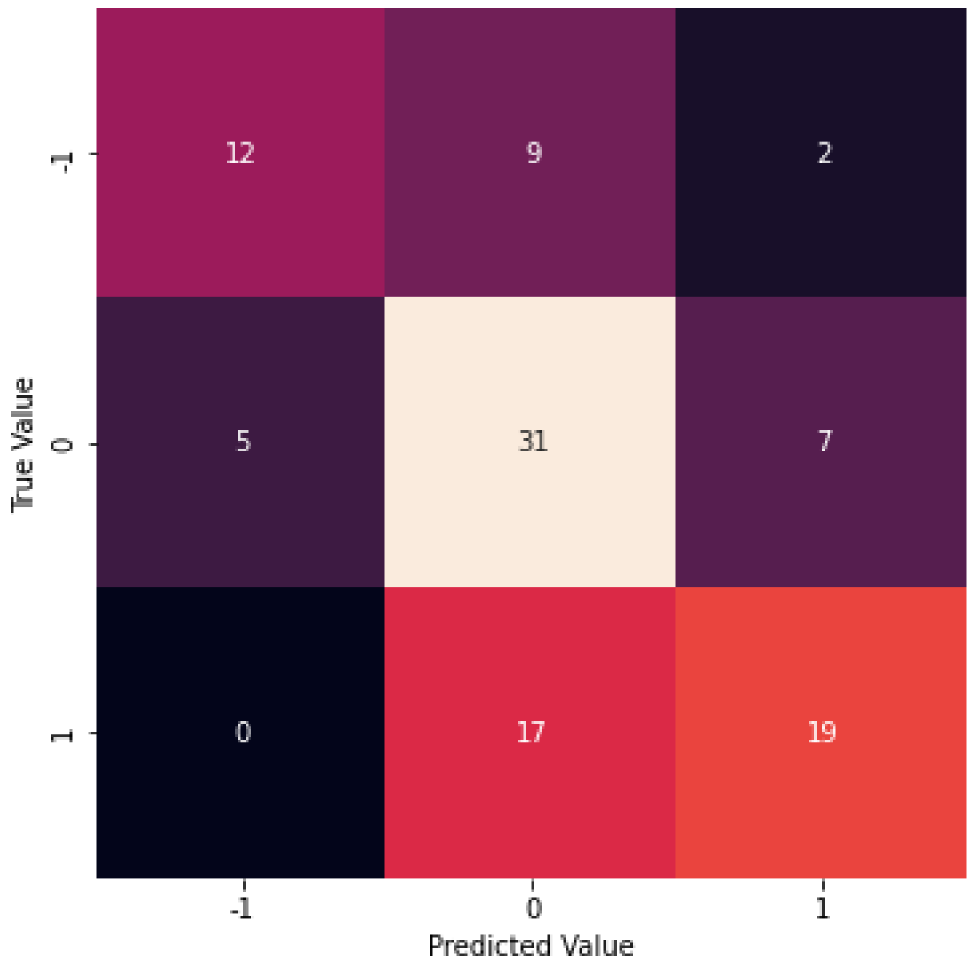
[76]:
```
kf = KFold(n_splits = 2)
temp_lst = []
clf_knn = KNeighborsClassifier(n_neighbors=4, weights='uniform')
for train_index, test_index in kf.split(influencer_x):
    X_train, X_test = influencer_x[train_index], influencer_x[test_index]
    y_train, y_test = influencer_y[train_index], influencer_y[test_index]
    clf_knn.fit(X_train, y_train)
    y_hat = clf_knn.predict(X_test)
    temp_lst2 = []
    temp_lst2.append(metrics.accuracy_score(y_train, clf_knn.predict(X_train)))
    temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
    temp_lst2.append(metrics.f1_score(y_test, y_hat, average='micro'))
    temp_lst2.append(metrics.jaccard_score(y_test, y_hat, average='micro'))
    temp_lst2.append(y_test)
    temp_lst2.append(y_hat)
    temp_lst2.append(X_test)
    temp_lst.append(temp_lst2)
```

[77]:
```
temp_lst_ytest, temp_lst_yhat, temp_lst_xtest = [], [], []
for row in temp_lst:
    for i in row[4]:
        temp_lst_ytest.append(i)
    for j in row[5]:
        temp_lst_yhat.append(j)
    for k in row[6]:
        temp_lst_xtest.append(k)

cnf_ytest = np.array(temp_lst_ytest)
cnf_yhat = np.array(temp_lst_yhat)
cnf_xtest = np.array(temp_lst_xtest)
print(metrics.classification_report(cnf_ytest, cnf_yhat))
```

```
              precision    recall  f1-score   support

          -1       0.71      0.52      0.60        23
           0       0.54      0.72      0.62        43
           1       0.68      0.53      0.59        36

    accuracy                           0.61       102
   macro avg       0.64      0.59      0.60       102
weighted avg       0.63      0.61      0.61       102
```

[78]:
```
cnf_matrix = confusion_matrix(cnf_ytest, cnf_yhat)
plt.figure(figsize=(8,6))
sns.heatmap(cnf_matrix, square=True, annot=True, cbar=False)
plt.xticks([.5, 1.5, 2.5], [-1, 0, 1])
plt.yticks([.5, 1.5, 2.5], [-1, 0, 1])
```

```
plt.xlabel('Predicted Value')
plt.ylabel('True Value')
plt.show()
```

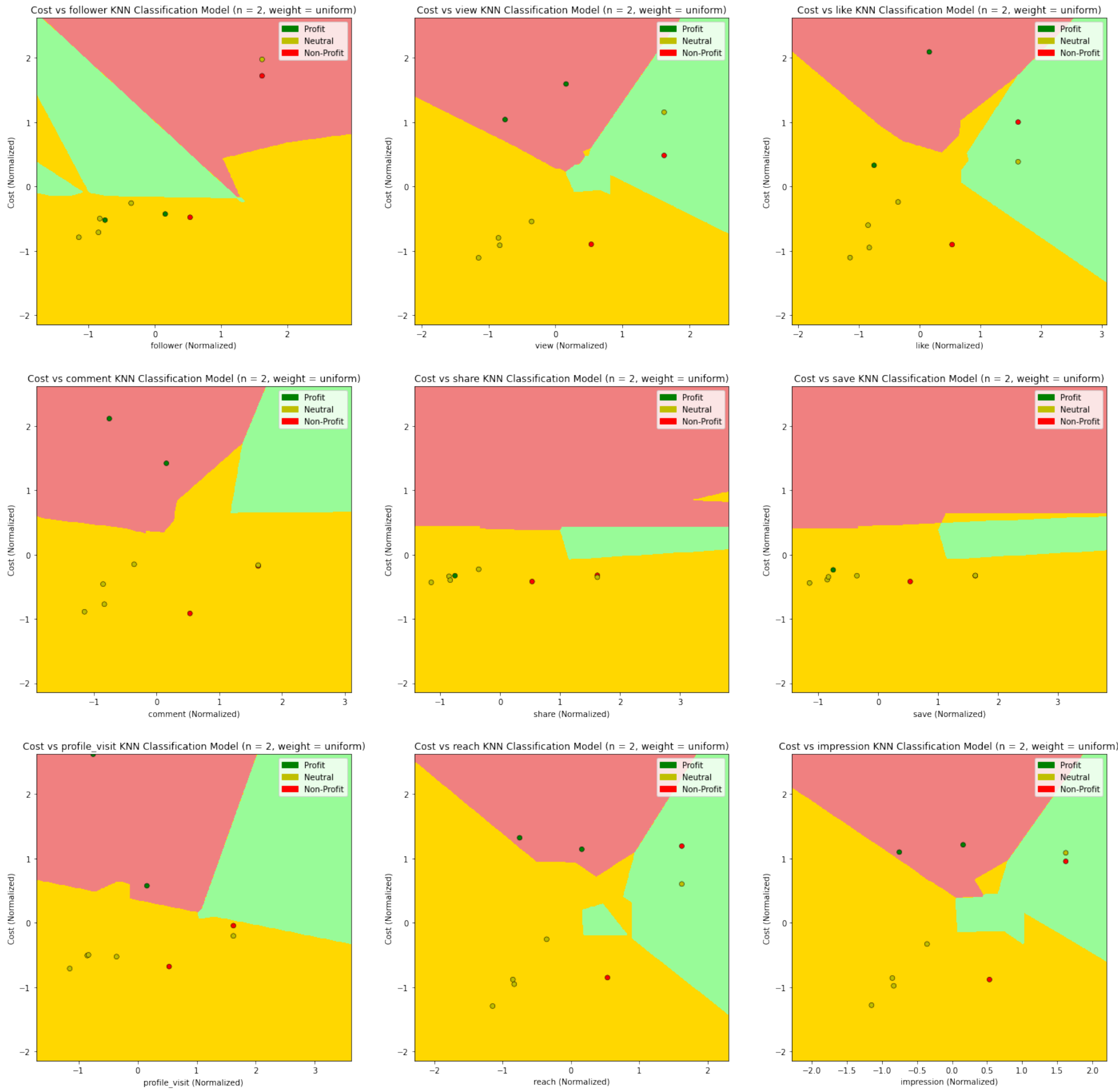


[79]:
```
fig = plt.figure(figsize = (24, 24))
ax1 = fig.add_subplot(3,3,1)
ax2 = fig.add_subplot(3,3,2)
ax3 = fig.add_subplot(3,3,3)
ax4 = fig.add_subplot(3,3,4)
ax5 = fig.add_subplot(3,3,5)
ax6 = fig.add_subplot(3,3,6)
ax7 = fig.add_subplot(3,3,7)
ax8 = fig.add_subplot(3,3,8)
ax9 = fig.add_subplot(3,3,9)
```

```
X,y = leaders_post_x[:,:10], leaders_post_y
h = .01
cmap_light = ListedColormap(['lightcoral', 'gold', 'palegreen'])
cmap_bold = ['r', 'y', 'g']
X_follower = X[:,(0,9)]
X_view = X[:,(1,9)]
X_like = X[:,(2,9)]
X_comment = X[:,(3,9)]
X_share = X[:,(4,9)]
X_save = X[:,(5,9)]
X_profile_visit = X[:,(6,9)]
X_reach = X[:,(7,9)]
X_impression = X[:,(8,9)]


xx_follower, xx_view, xx_like, xx_comment, xx_share, xx_save, xx_profile_visit,␣
 ↪xx_reach, xx_impression = None, None, None, None, None, None, None, None,␣
 ↪None
yy_follower, yy_view, yy_like, yy_comment, yy_share, yy_save, yy_profile_visit,␣
 ↪yy_reach, yy_impression = None, None, None, None, None, None, None, None,␣
 ↪None
Z_follower, Z_view, Z_like, Z_comment, Z_share, Z_save, Z_profile_visit,␣
 ↪Z_reach, Z_impression = None, None, None, None, None, None, None, None, None
g1, g2, g3, g4, g5, g6, g7, g8, g9 = None, None, None, None, None, None, None,␣
 ↪None, None

X_lst = [X_follower, X_view, X_like, X_comment, X_share, X_save,␣
 ↪X_profile_visit, X_reach, X_impression]
xx_lst = [xx_follower, xx_view, xx_like, xx_comment, xx_share, xx_save,␣
 ↪xx_profile_visit, xx_reach, xx_impression]
yy_lst = [yy_follower, yy_view, yy_like, yy_comment, yy_share, yy_save,␣
 ↪yy_profile_visit, yy_reach, yy_impression]
Z_lst = [Z_follower, Z_view, Z_like, Z_comment, Z_share, Z_save,␣
 ↪Z_profile_visit, Z_reach, Z_impression]
ax_lst = [ax1, ax2, ax3, ax4, ax5, ax6, ax7, ax8, ax9]
g_lst = [g1, g2, g3, g4, g5, g6, g7, g8, g9]
x_label_lst = ['follower', 'view', 'like', 'comment', 'share', 'save',␣
 ↪'profile_visit', 'reach', 'impression']
labels=['Profit','Neutral', 'Non-Profit']
red_patch = patches.Patch(color='r', label='Non-Profit')
yellow_patch = patches.Patch(color='y', label='Neutral')
green_patch = patches.Patch(color='g', label='Profit')


def plot_calc(x, y = y):
```

```
    '''
    This function is for calculating the area to plot with colors according to␣
 ↪the input
        input -> x and y.
        return -> xx, yy, Z which are needed to drawing the contour and plot.
    '''
    clf_knn.fit(x, y)
    x_min, x_max = x[:, 0].min() - 1, x[:, 0].max() + 1
    y_min, y_max = x[:, 1].min() - 1, x[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = clf_knn.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    return xx, yy, Z

for ax_, xx_, yy_, Z_, X_, x_label_, g_ in tqdm(zip(ax_lst, xx_lst, yy_lst,␣
 ↪Z_lst, X_lst, x_label_lst, g_lst)):
    xx_, yy_, Z_ = plot_calc(X_, y)
    ax_.contourf(xx_, yy_, Z_, cmap=cmap_light)
    g_ = sns.scatterplot(x=X_[:, 1], y=X_[:, 0], hue=leaders_post['benefit'],␣
 ↪palette=cmap_bold, alpha=1.0, edgecolor='black', ax=ax_)
    ax_.set_title(f'Cost vs {x_label_} KNN Classification Model (n = 2, weight␣
 ↪= uniform)')
    ax_.set_xlim(xx_.min(), xx_.max())
    ax_.set_ylim(yy_.min(), yy_.max())
    ax_.set_xlabel(f'{x_label_} (Normalized)')
    ax_.set_ylabel('Cost (Normalized)')
    ax_.legend(handles=[green_patch, yellow_patch, red_patch],loc = 'upper␣
 ↪right', fontsize = 10);

plt.show()
```

```
9it [01:04,  7.11s/it]
```



1.1.4 Decision Tree

Advertising Posts

[80]:
```python
from sklearn.tree import DecisionTreeClassifier, plot_tree
from matplotlib.colors import ListedColormap
```

[81]:
```python
criterion = ['gini', 'entropy']
```

[82]:
```python
temp_lst = []
for i in tqdm_notebook(range(2, 9)):
    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 criterion:
            dtc = DecisionTreeClassifier(criterion = c)
            dtc.fit(X_train, y_train)
            temp_lst2 = []
            temp_lst2.append(i)
            temp_lst2.append(c)
            temp_lst2.append(dtc.score(X_train, y_train))
            temp_lst2.append(dtc.score(X_test, y_test))
            temp_lst.append(temp_lst2)

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

temp_lst = []
for k in range(2, 9):
    for c_ in criterion:
        temp_lst2 = []
        temp_lst2.append(k)
        temp_lst2.append(c_)
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['Criterion'] == c_)]['DTC Train Score']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
 ↪(temp_df['Criterion'] == c_)]['DTC Test Score']), decimals=4))
        temp_lst.append(temp_lst2)

dt_clf_eval_df = pd.DataFrame(temp_lst, columns=['k', 'Criterion', 'DTC Train␣
 ↪Score', 'DTC Test Score'])
dt_clf_eval_df
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=7.0),␣
 ↪HTML(value='')))
```

[82]:

	k	Criterion	DTC Train Score	DTC Test Score
0	2	gini	1.00	0.67
1	2	entropy	1.00	0.67
2	3	gini	1.00	0.70
3	3	entropy	1.00	0.67
4	4	gini	1.00	0.69
5	4	entropy	1.00	0.64
6	5	gini	1.00	0.69
7	5	entropy	1.00	0.58
8	6	gini	1.00	0.72
9	6	entropy	1.00	0.68
10	7	gini	1.00	0.55
11	7	entropy	1.00	0.58
12	8	gini	1.00	0.61

```
13  8   entropy            1.00           0.68
```

[83]:
```
dt_clf_eval_df.nlargest(3, 'DTC Test Score')
```

[83]:

	k	Criterion	DTC Train Score	DTC Test Score
8	6	gini	1.00	0.72
2	3	gini	1.00	0.70
6	5	gini	1.00	0.69

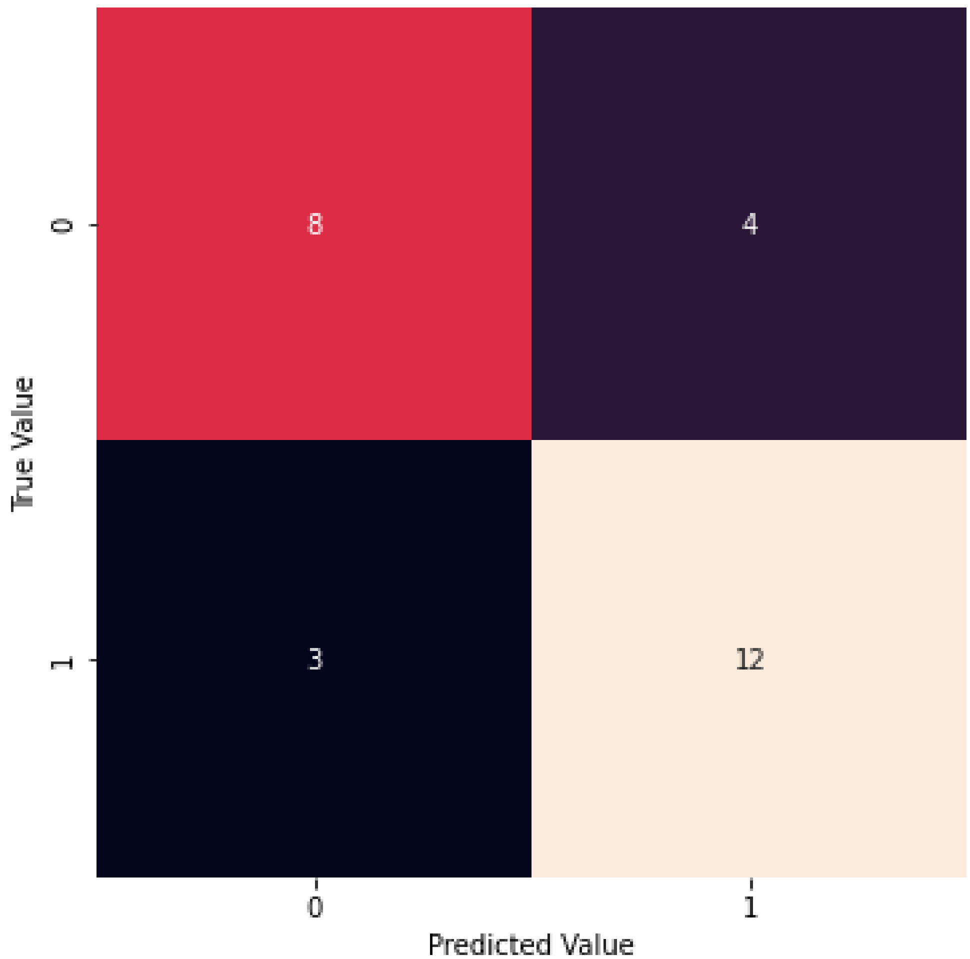
[84]:
```
kf = KFold(n_splits = 3)
temp_lst = []
clf_dt = DecisionTreeClassifier(criterion = 'entropy')
for train_index, test_index in kf.split(ad_post_x):
    X_train, X_test = ad_post_x[train_index], ad_post_x[test_index]
    y_train, y_test = ad_post_y[train_index], ad_post_y[test_index]
    clf_dt.fit(X_train, y_train)
    y_hat = clf_dt.predict(X_test)
    y_hat_prob = clf_dt.predict_proba(X_test)
    temp_lst2 = []
    temp_lst2.append(metrics.accuracy_score(y_train, clf_dt.predict(X_train)))
    temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
    temp_lst2.append(metrics.f1_score(y_test, y_hat))
    temp_lst2.append(metrics.jaccard_score(y_test, y_hat))
    temp_lst2.append(metrics.log_loss(y_test, y_hat_prob))
    temp_lst2.append(y_test)
    temp_lst2.append(y_hat)
    temp_lst2.append(X_test)
    temp_lst.append(temp_lst2)
```

[85]:
```
temp_lst_ytest, temp_lst_yhat, temp_lst_xtest = [], [], []
for row in temp_lst:
    for i in row[5]:
        temp_lst_ytest.append(i)
    for j in row[6]:
        temp_lst_yhat.append(j)
    for k in row[7]:
        temp_lst_xtest.append(k)

cnf_ytest = np.array(temp_lst_ytest)
cnf_yhat = np.array(temp_lst_yhat)
cnf_xtest = np.array(temp_lst_xtest)
print(metrics.classification_report(cnf_ytest, cnf_yhat))
```

```
              precision    recall  f1-score   support

           0       0.73      0.67      0.70        12
           1       0.75      0.80      0.77        15
```

```
    accuracy                           0.74        27
   macro avg       0.74      0.73      0.73        27
weighted avg       0.74      0.74      0.74        27
```

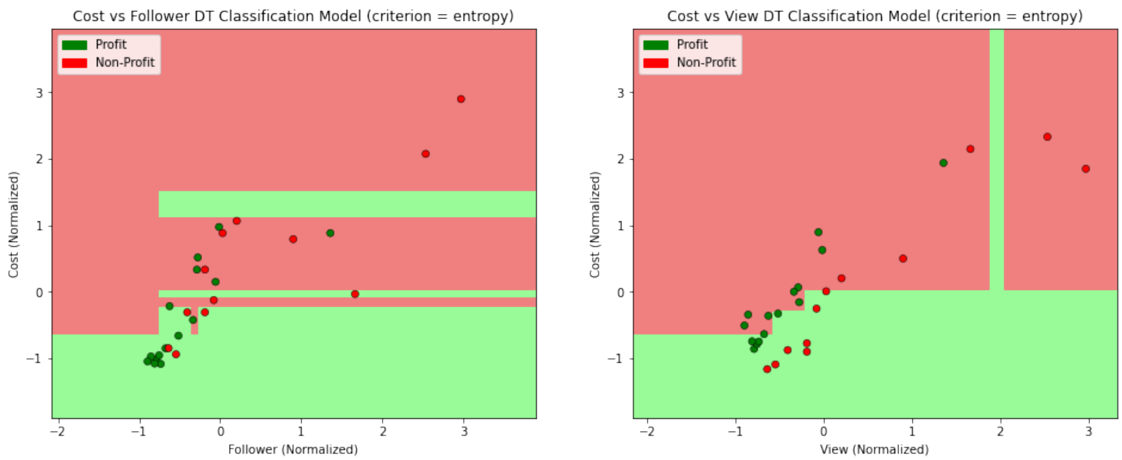
[86]:
```
cnf_matrix = confusion_matrix(cnf_ytest, cnf_yhat)
plt.figure(figsize=(8,6))
sns.heatmap(cnf_matrix, square=True, annot=True, cbar=False)
# plt.xticks([.5, 1.5, 2.5], [-1, 0, 1])
# plt.yticks([.5, 1.5, 2.5], [-1, 0, 1])
plt.xlabel('Predicted Value')
plt.ylabel('True Value')
plt.show()
```

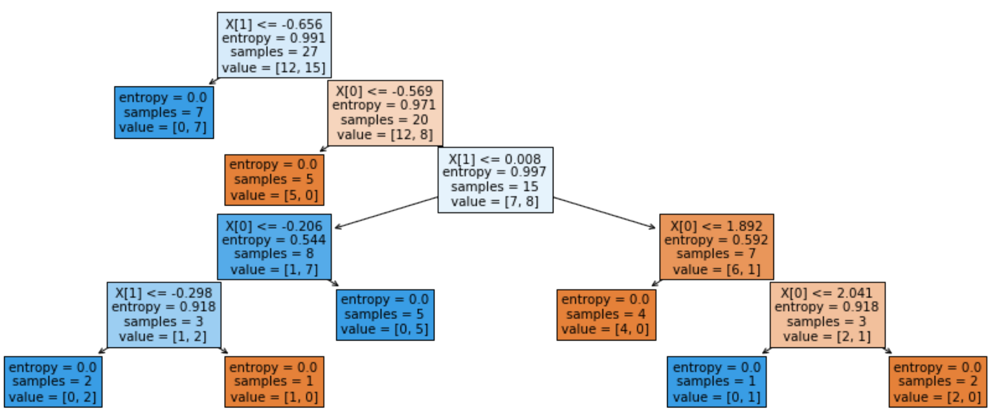


[87]:
```
fig = plt.figure(figsize = (16, 6))
ax1 = fig.add_subplot(1,2,1)
ax2 = fig.add_subplot(1,2,2)
```

```python
X,y = ad_post_x[:,:3], ad_post_y
h = .01
cmap_light = ListedColormap(['lightcoral', 'palegreen'])
cmap_bold = ['r', 'g']
X_follower = X[:,(0,2)]
X_view = X[:,(1,2)]

xx_follower, xx_view = None, None
yy_follower, yy_view = None, None
Z_follower, Z_view = None, None
g1, g2 = None, None

X_lst = [X_follower, X_view]
xx_lst = [xx_follower, xx_view]
yy_lst = [yy_follower, yy_view]
Z_lst = [Z_follower, Z_view]
ax_lst = [ax1, ax2]
g_lst = [g1, g2]
x_label_lst = ['Follower', 'View']
labels=['Non-Profit','Profit']
red_patch = patches.Patch(color='r', label='Non-Profit')
green_patch = patches.Patch(color='g', label='Profit')


def plot_calc(x, y = y):
    '''
    This function is for calculating the area to plot with colors according to␣
↪the input
        input -> x and y.
        return -> xx, yy, Z which are needed to drawing the contour and plot.
    '''
    clf_dt.fit(x, y)
    x_min, x_max = x[:, 0].min() - 1, x[:, 0].max() + 1
    y_min, y_max = x[:, 1].min() - 1, x[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = clf_dt.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    return xx, yy, Z

for ax_, xx_, yy_, Z_, X_, x_label_, g_ in zip(ax_lst, xx_lst, yy_lst, Z_lst,␣
↪X_lst, x_label_lst, g_lst):
    xx_, yy_, Z_ = plot_calc(X_, y)
    ax_.contourf(xx_, yy_, Z_, cmap=cmap_light)
    g_ = sns.scatterplot(x=X_[:, 1], y=X_[:, 0], hue=ad_post['benefit'],␣
↪palette=cmap_bold, alpha=1.0, edgecolor='black', ax=ax_)
```

```
    ax_.set_title(f'Cost vs {x_label_} DT Classification Model (criterion =␣
↪entropy)')
    ax_.set_xlim(xx_.min(), xx_.max())
    ax_.set_ylim(yy_.min(), yy_.max())
    ax_.set_xlabel(f'{x_label_} (Normalized)')
    ax_.set_ylabel('Cost (Normalized)')
    ax_.legend(handles=[green_patch, red_patch],loc = 'upper left', fontsize =␣
↪10);

plt.show()
plt.figure(figsize = (16, 6))
plot_tree(clf_dt, filled=True)
plt.show()
```





Advertising Story

[88]:

```python
temp_lst = []
for i in tqdm_notebook(range(2, 9)):
    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 criterion:
            dtc = DecisionTreeClassifier(criterion = c)
            dtc.fit(X_train, y_train)
            temp_lst2 = []
            temp_lst2.append(i)
            temp_lst2.append(c)
            temp_lst2.append(dtc.score(X_train, y_train))
            temp_lst2.append(dtc.score(X_test, y_test))
            temp_lst.append(temp_lst2)

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

temp_lst = []
for k in range(2, 9):
    for c_ in criterion:
        temp_lst2 = []
        temp_lst2.append(k)
        temp_lst2.append(c_)
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['Criterion'] == c_)]['DTC Train Score']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &␣
↪(temp_df['Criterion'] == c_)]['DTC Test Score']), decimals=4))
        temp_lst.append(temp_lst2)

dt_clf_eval_df = pd.DataFrame(temp_lst, columns=['k', 'Criterion', 'DTC Train␣
↪Score', 'DTC Test Score'])
dt_clf_eval_df
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=7.0),␣
↪HTML(value='')))
```

[88]:

	k	Criterion	DTC Train Score	DTC Test Score
0	2	gini	1.00	0.66
1	2	entropy	1.00	0.74
2	3	gini	1.00	0.59
3	3	entropy	1.00	0.70
4	4	gini	1.00	0.74
5	4	entropy	1.00	0.63
6	5	gini	1.00	0.71

```
7   5   entropy           1.00           0.63
8   6      gini           1.00           0.67
9   6   entropy           1.00           0.55
10  7      gini           1.00           0.70
11  7   entropy           1.00           0.80
12  8      gini           1.00           0.74
13  8   entropy           1.00           0.74
```

[89]:
```
dt_clf_eval_df.nlargest(3, 'DTC Test Score')
```

[89]:
```
    k Criterion  DTC Train Score  DTC Test Score
11  7   entropy             1.00            0.80
4   4      gini             1.00            0.74
12  8      gini             1.00            0.74
```

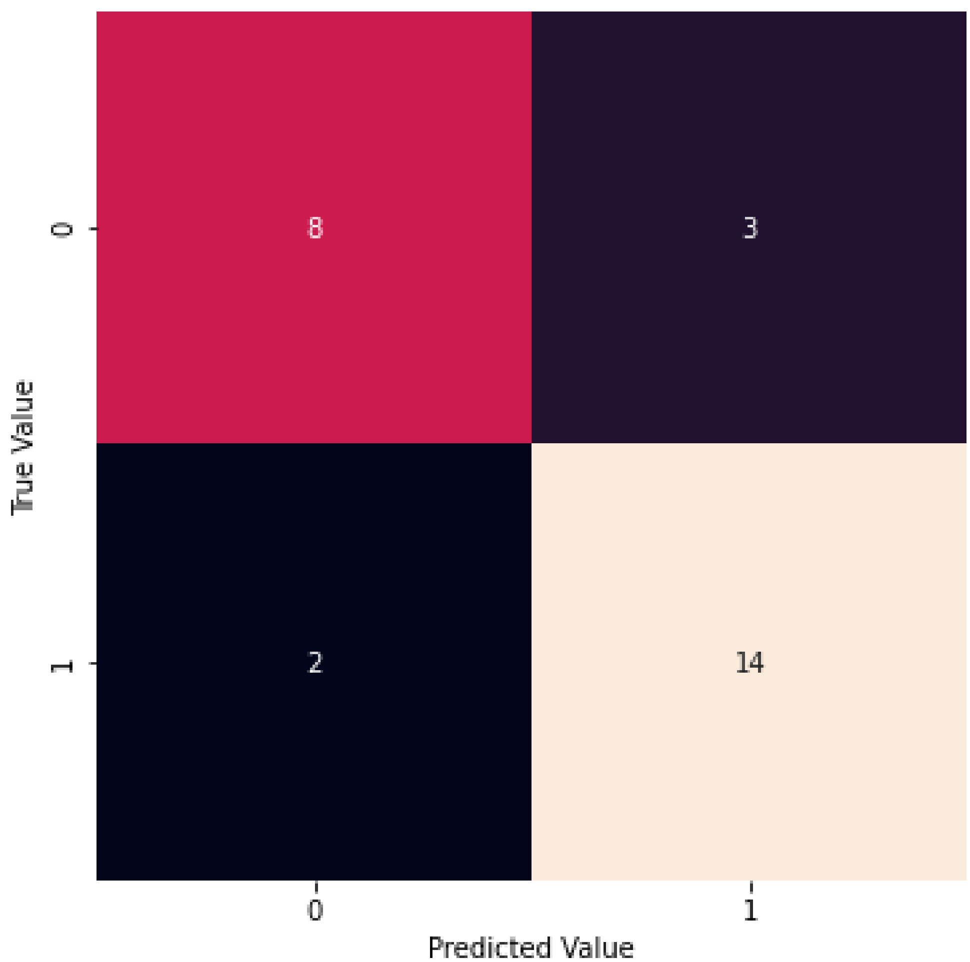
[90]:
```
kf = KFold(n_splits = 8)
temp_lst = []
clf_dt = DecisionTreeClassifier(criterion = 'entropy')
for train_index, test_index in kf.split(ad_story_x):
    X_train, X_test = ad_story_x[train_index], ad_story_x[test_index]
    y_train, y_test = ad_story_y[train_index], ad_story_y[test_index]
    clf_dt.fit(X_train, y_train)
    y_hat = clf_dt.predict(X_test)
    y_hat_prob = clf_dt.predict_proba(X_test)
    temp_lst2 = []
    temp_lst2.append(metrics.accuracy_score(y_train, clf_dt.predict(X_train)))
    temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
    temp_lst2.append(metrics.f1_score(y_test, y_hat))
    temp_lst2.append(metrics.jaccard_score(y_test, y_hat))
    temp_lst2.append(y_test)
    temp_lst2.append(y_hat)
    temp_lst2.append(X_test)
    temp_lst.append(temp_lst2)
```

[91]:
```
temp_lst_ytest, temp_lst_yhat, temp_lst_xtest = [], [], []
for row in temp_lst:
    for i in row[4]:
        temp_lst_ytest.append(i)
    for j in row[5]:
        temp_lst_yhat.append(j)
    for k in row[6]:
        temp_lst_xtest.append(k)

cnf_ytest = np.array(temp_lst_ytest)
cnf_yhat = np.array(temp_lst_yhat)
cnf_xtest = np.array(temp_lst_xtest)
print(metrics.classification_report(cnf_ytest, cnf_yhat))
```

```
              precision    recall  f1-score   support

           0       0.80      0.73      0.76        11
           1       0.82      0.88      0.85        16

    accuracy                           0.81        27
   macro avg       0.81      0.80      0.81        27
weighted avg       0.81      0.81      0.81        27
```

[92]:

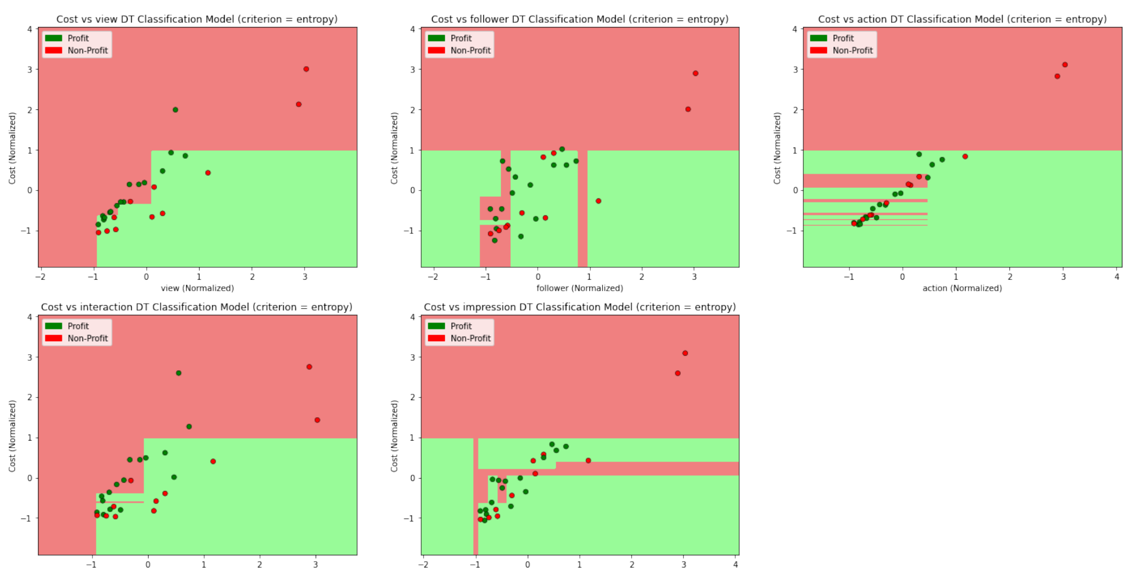
```
cnf_matrix = confusion_matrix(cnf_ytest, cnf_yhat)
plt.figure(figsize=(8,6))
sns.heatmap(cnf_matrix, square=True, annot=True, cbar=False)
# plt.xticks([.5, 1.5, 2.5], [-1, 0, 1])
# plt.yticks([.5, 1.5, 2.5], [-1, 0, 1])
plt.xlabel('Predicted Value')
plt.ylabel('True Value')
plt.show()
```

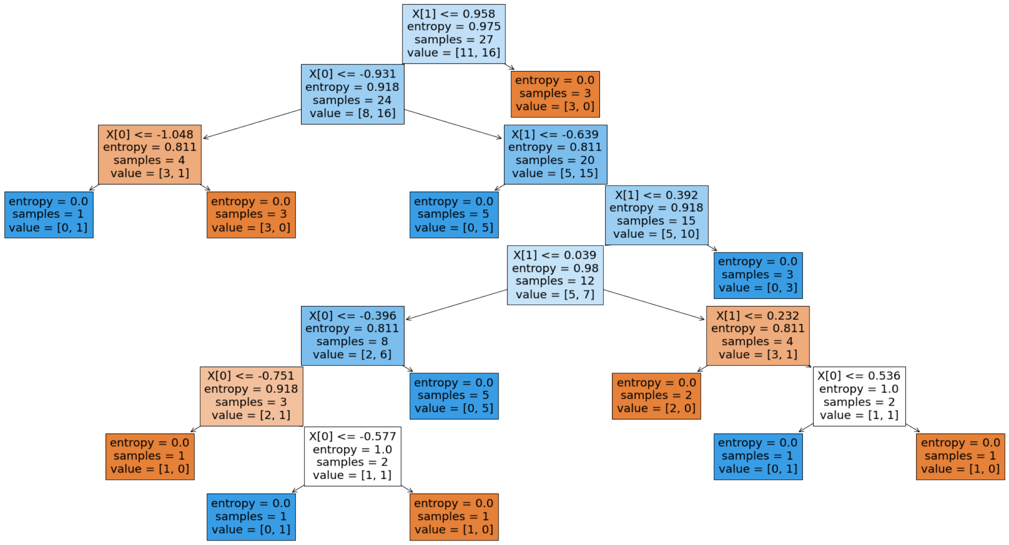


[93]:

```
fig = plt.figure(figsize = (24, 12))
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)


X,y = ad_story_x[:,:6], ad_story_y
h = .01
cmap_light = ListedColormap(['lightcoral', 'palegreen'])
cmap_bold = ['r', 'g']
X_view = X[:,(0,5)]
X_follower = X[:,(1,5)]
X_action = X[:,(2,5)]
X_interaction = X[:,(3,5)]
X_impression = X[:,(4,5)]


xx_view, xx_follower, xx_action, xx_interaction, xx_impression = None, None,␣
↪None, None, None
yy_view, yy_follower, yy_action, yy_interaction, yy_impression = None, None,␣
↪None, None, None
Z_view, Z_follower, Z_action, Z_interaction, Z_impression = None, None, None,␣
↪None, None
g1, g2, g3, g4, g5 = None, None, None, None, None

X_lst = [X_view, X_follower, X_action, X_interaction, X_impression]
xx_lst = [xx_view, xx_follower, xx_action, xx_interaction, xx_impression]
yy_lst = [yy_view, yy_follower, yy_action, yy_interaction, yy_impression]
Z_lst = [Z_view, Z_follower, Z_action, Z_interaction, Z_impression]
ax_lst = [ax1, ax2, ax3, ax4, ax5]
g_lst = [g1, g2, g3, g4, g5]
x_label_lst = ['view', 'follower', 'action', 'interaction', 'impression']
labels=['Non-Profit','Profit']
red_patch = patches.Patch(color='r', label='Non-Profit')
green_patch = patches.Patch(color='g', label='Profit')


def plot_calc(x, y = y):
    '''
    This function is for calculating the area to plot with colors according to␣
↪the input
        input -> x and y.
        return -> xx, yy, Z which are needed to drawing the contour and plot.
```

```
    '''
    clf_dt.fit(x, y)
    x_min, x_max = x[:, 0].min() - 1, x[:, 0].max() + 1
    y_min, y_max = x[:, 1].min() - 1, x[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = clf_dt.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    return xx, yy, Z

for ax_, xx_, yy_, Z_, X_, x_label_, g_ in zip(ax_lst, xx_lst, yy_lst, Z_lst, X_lst, x_label_lst, g_lst):
    xx_, yy_, Z_ = plot_calc(X_, y)
    ax_.contourf(xx_, yy_, Z_, cmap=cmap_light)
    g_ = sns.scatterplot(x=X_[:, 1], y=X_[:, 0], hue=ad_story['benefit'], palette=cmap_bold, alpha=1.0, edgecolor='black', ax=ax_)
    ax_.set_title(f'Cost vs {x_label_} DT Classification Model (criterion = entropy)')
    ax_.set_xlim(xx_.min(), xx_.max())
    ax_.set_ylim(yy_.min(), yy_.max())
    ax_.set_xlabel(f'{x_label_} (Normalized)')
    ax_.set_ylabel('Cost (Normalized)')
    ax_.legend(handles=[green_patch, red_patch],loc = 'upper left', fontsize = 10);

plt.show()
plt.figure(figsize = (32, 16))
plot_tree(clf_dt, filled=True)
plt.show()
```





Influencer

[94]:
```python
temp_lst = []
for i in tqdm_notebook(range(2, 9)):
    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 criterion:
            dtc = DecisionTreeClassifier(criterion = c)
            dtc.fit(X_train, y_train)
            temp_lst2 = []
            temp_lst2.append(i)
            temp_lst2.append(c)
            temp_lst2.append(dtc.score(X_train, y_train))
            temp_lst2.append(dtc.score(X_test, y_test))
            temp_lst.append(temp_lst2)

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

temp_lst = []
for k in range(2, 9):
    for c_ in criterion:
        temp_lst2 = []
        temp_lst2.append(k)
        temp_lst2.append(c_)
```

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

dt_clf_eval_df = pd.DataFrame(temp_lst, columns=['k', 'Criterion', 'DTC Train␣
↪Score', 'DTC Test Score'])
dt_clf_eval_df
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=7.0),␣
↪HTML(value='')))
```

[94]:

	k	Criterion	DTC Train Score	DTC Test Score
0	2	gini	1.00	0.49
1	2	entropy	1.00	0.47
2	3	gini	1.00	0.46
3	3	entropy	1.00	0.41
4	4	gini	1.00	0.54
5	4	entropy	1.00	0.54
6	5	gini	1.00	0.65
7	5	entropy	1.00	0.50
8	6	gini	1.00	0.73
9	6	entropy	1.00	0.72
10	7	gini	1.00	0.58
11	7	entropy	1.00	0.73
12	8	gini	1.00	0.71
13	8	entropy	1.00	0.78

[95]:
```
dt_clf_eval_df.nlargest(3, 'DTC Test Score')
```

[95]:

	k	Criterion	DTC Train Score	DTC Test Score
13	8	entropy	1.00	0.78
11	7	entropy	1.00	0.73
8	6	gini	1.00	0.73

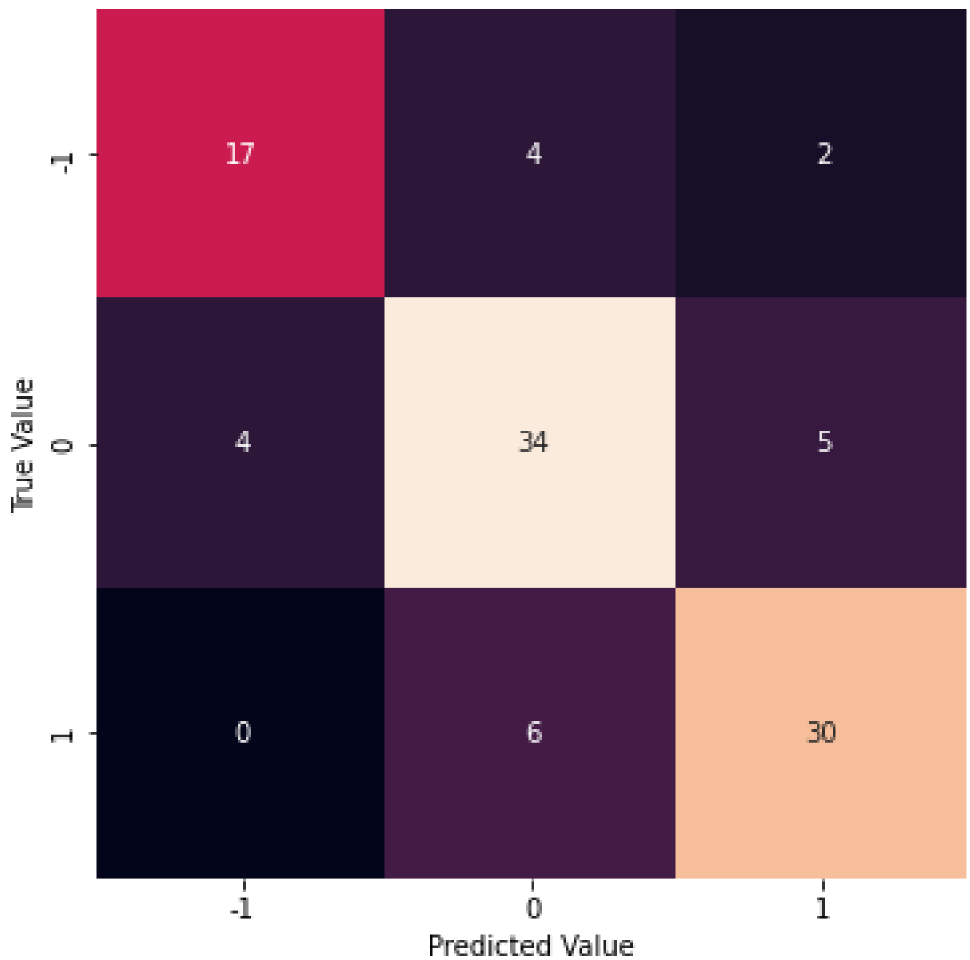
[96]:
```
kf = KFold(n_splits = 8)
temp_lst = []
clf_dt = DecisionTreeClassifier(criterion = 'entropy')
for train_index, test_index in kf.split(influencer_x):
    X_train, X_test = influencer_x[train_index], influencer_x[test_index]
    y_train, y_test = influencer_y[train_index], influencer_y[test_index]
    clf_dt.fit(X_train, y_train)
    y_hat = clf_dt.predict(X_test)
    y_hat_prob = clf_dt.predict_proba(X_test)
    temp_lst2 = []
```

```
    temp_lst2.append(metrics.accuracy_score(y_train, clf_dt.predict(X_train)))
    temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
    temp_lst2.append(metrics.f1_score(y_test, y_hat, average='micro'))
    temp_lst2.append(metrics.jaccard_score(y_test, y_hat, average='micro'))
    temp_lst2.append(y_test)
    temp_lst2.append(y_hat)
    temp_lst2.append(X_test)
    temp_lst.append(temp_lst2)
```

[97]:
```
temp_lst_ytest, temp_lst_yhat, temp_lst_xtest = [], [], []
for row in temp_lst:
    for i in row[4]:
        temp_lst_ytest.append(i)
    for j in row[5]:
        temp_lst_yhat.append(j)
    for k in row[6]:
        temp_lst_xtest.append(k)

cnf_ytest = np.array(temp_lst_ytest)
cnf_yhat = np.array(temp_lst_yhat)
cnf_xtest = np.array(temp_lst_xtest)
print(metrics.classification_report(cnf_ytest, cnf_yhat))
```

```
              precision    recall  f1-score   support

          -1       0.81      0.74      0.77        23
           0       0.77      0.79      0.78        43
           1       0.81      0.83      0.82        36

    accuracy                           0.79       102
   macro avg       0.80      0.79      0.79       102
weighted avg       0.79      0.79      0.79       102
```

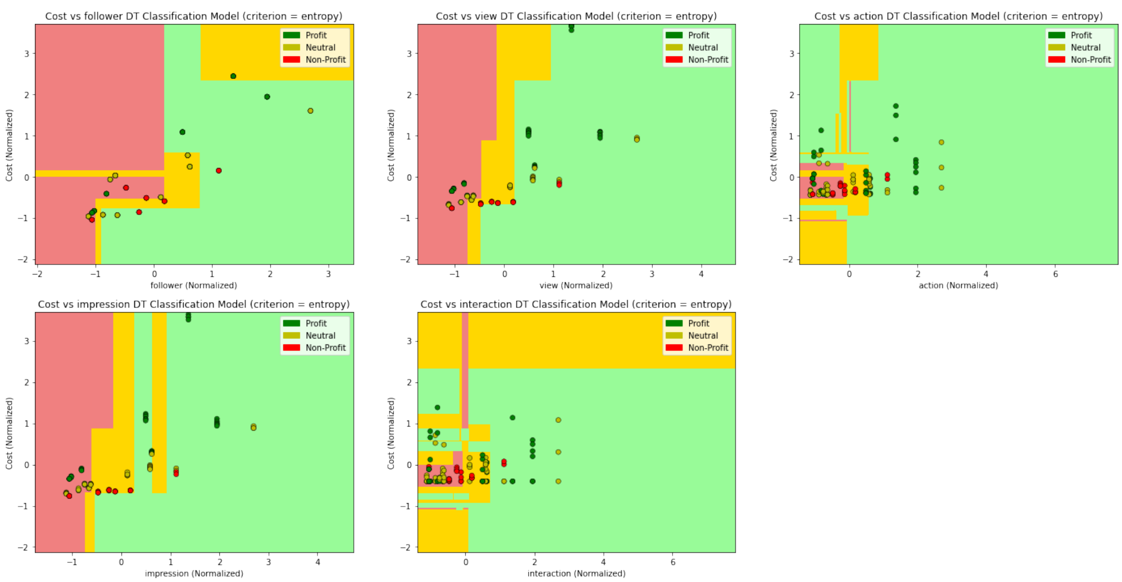
[98]:
```
cnf_matrix = confusion_matrix(cnf_ytest, cnf_yhat)
plt.figure(figsize=(8,6))
sns.heatmap(cnf_matrix, square=True, annot=True, cbar=False)
plt.xticks([.5, 1.5, 2.5], [-1, 0, 1])
plt.yticks([.5, 1.5, 2.5], [-1, 0, 1])
plt.xlabel('Predicted Value')
plt.ylabel('True Value')
plt.show()
```

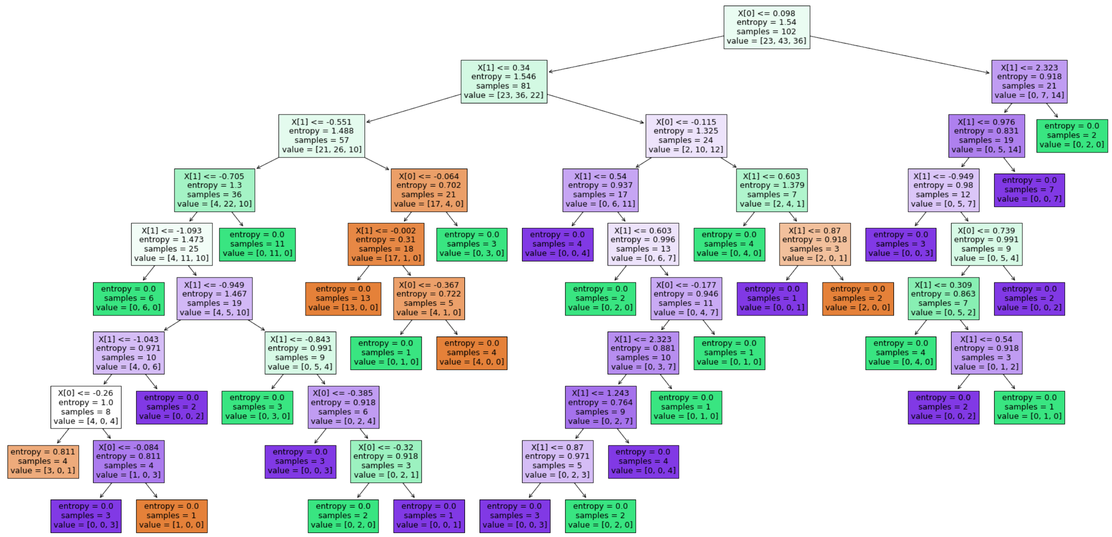


[99]:
```
fig = plt.figure(figsize = (24, 12))
ax1 = fig.add_subplot(2,3,1)
ax2 = fig.add_subplot(2,3,2)
ax3 = fig.add_subplot(2,3,3)
ax4 = fig.add_subplot(2,3,4)
ax5 = fig.add_subplot(2,3,5)


X,y = influencer_x[:,:7], influencer_y
h = .01
cmap_light = ListedColormap(['lightcoral', 'gold', 'palegreen'])
cmap_bold = ['r', 'y', 'g']
X_follower = X[:,(0,6)]
X_view = X[:,(1,6)]
X_action = X[:,(2,6)]
X_impression = X[:,(3,6)]
```

```
X_interaction = X[:,(5,6)]


xx_follower, xx_view, xx_action, xx_impression, xx_interaction = None, None,␣
 ↪None, None, None
yy_follower, yy_view, yy_action, yy_impression, yy_interaction = None, None,␣
 ↪None, None, None
Z_follower, Z_view, Z_action, Z_impression, Z_interaction = None, None, None,␣
 ↪None, None
g1, g2, g3, g4, g5 = None, None, None, None, None

X_lst = [X_follower, X_view, X_action, X_impression, X_interaction]
xx_lst = [xx_follower, xx_view, xx_action, xx_impression, xx_interaction]
yy_lst = [yy_follower, yy_view, yy_action, yy_impression, yy_interaction]
Z_lst = [Z_follower, Z_view, Z_action, Z_impression, Z_interaction]
ax_lst = [ax1, ax2, ax3, ax4, ax5]
g_lst = [g1, g2, g3, g4, g5]
x_label_lst = ['follower', 'view', 'action', 'impression', 'interaction']
labels=['Non-Profit','Neutral', 'Profit']
red_patch = patches.Patch(color='r', label='Non-Profit')
yellow_patch = patches.Patch(color='y', label='Neutral')
green_patch = patches.Patch(color='g', label='Profit')


def plot_calc(x, y = y):
    '''
    This function is for calculating the area to plot with colors according to␣
 ↪the input
        input -> x and y.
        return -> xx, yy, Z which are needed to drawing the contour and plot.
    '''
    clf_dt.fit(x, y)
    x_min, x_max = x[:, 0].min() - 1, x[:, 0].max() + 1
    y_min, y_max = x[:, 1].min() - 1, x[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = clf_dt.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    return xx, yy, Z

for ax_, xx_, yy_, Z_, X_, x_label_, g_ in zip(ax_lst, xx_lst, yy_lst, Z_lst,␣
 ↪X_lst, x_label_lst, g_lst):
    xx_, yy_, Z_ = plot_calc(X_, y)
    ax_.contourf(xx_, yy_, Z_, cmap=cmap_light)
    g_ = sns.scatterplot(x=X_[:, 1], y=X_[:, 0], hue=influencer['benefit'],␣
 ↪palette=cmap_bold, alpha=1.0, edgecolor='black', ax=ax_)
```

```
    ax_.set_title(f'Cost vs {x_label_} DT Classification Model (criterion =␣
↪entropy)')
    ax_.set_xlim(xx_.min(), xx_.max())
    ax_.set_ylim(yy_.min(), yy_.max())
    ax_.set_xlabel(f'{x_label_} (Normalized)')
    ax_.set_ylabel('Cost (Normalized)')
    ax_.legend(handles=[green_patch, yellow_patch, red_patch],loc = 'upper␣
↪right', fontsize = 10);

plt.show()
plt.figure(figsize = (32, 16))
plot_tree(clf_dt, filled=True)
plt.show()
```





Leaders Post

[100]:
```python
temp_lst = []
for i in tqdm_notebook(range(2, 9)):
    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 criterion:
            dtc = DecisionTreeClassifier(criterion = c)
            dtc.fit(X_train, y_train)
            temp_lst2 = []
            temp_lst2.append(i)
            temp_lst2.append(c)
            temp_lst2.append(dtc.score(X_train, y_train))
            temp_lst2.append(dtc.score(X_test, y_test))
            temp_lst.append(temp_lst2)

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

temp_lst = []
for k in range(2, 9):
    for c_ in criterion:
        temp_lst2 = []
        temp_lst2.append(k)
        temp_lst2.append(c_)
```

```
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &
 ↪(temp_df['Criterion'] == c_)]['DTC Train Score']), decimals=4))
        temp_lst2.append(np.round(np.mean(temp_df[(temp_df['k'] == k) &
 ↪(temp_df['Criterion'] == c_)]['DTC Test Score']), decimals=4))
        temp_lst.append(temp_lst2)

dt_clf_eval_df = pd.DataFrame(temp_lst, columns=['k', 'Criterion', 'DTC Train
 ↪Score', 'DTC Test Score'])
dt_clf_eval_df
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=7.0),
 ↪HTML(value='')))
```

[100]:

	k	Criterion	DTC Train Score	DTC Test Score
0	2	gini	1.00	0.35
1	2	entropy	1.00	0.45
2	3	gini	1.00	0.44
3	3	entropy	1.00	0.56
4	4	gini	1.00	0.54
5	4	entropy	1.00	0.46
6	5	gini	1.00	0.40
7	5	entropy	1.00	0.40
8	6	gini	1.00	0.50
9	6	entropy	1.00	0.50
10	7	gini	1.00	0.36
11	7	entropy	1.00	0.36
12	8	gini	1.00	0.44
13	8	entropy	1.00	0.44

[101]:

```
dt_clf_eval_df.nlargest(3, 'DTC Test Score')
```

[101]:

	k	Criterion	DTC Train Score	DTC Test Score
3	3	entropy	1.00	0.56
4	4	gini	1.00	0.54
8	6	gini	1.00	0.50

[102]:

```
kf = KFold(n_splits = 8)
temp_lst = []
clf_dt = DecisionTreeClassifier(criterion = 'entropy')
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]
    clf_dt.fit(X_train, y_train)
    y_hat = clf_dt.predict(X_test)
    y_hat_prob = clf_dt.predict_proba(X_test)
    temp_lst2 = []
```

```
    temp_lst2.append(metrics.accuracy_score(y_train, clf_dt.predict(X_train)))
    temp_lst2.append(metrics.accuracy_score(y_test, y_hat))
    temp_lst2.append(metrics.f1_score(y_test, y_hat, average='micro'))
    temp_lst2.append(metrics.jaccard_score(y_test, y_hat, average='micro'))
    temp_lst2.append(y_test)
    temp_lst2.append(y_hat)
    temp_lst2.append(X_test)
    temp_lst.append(temp_lst2)
```

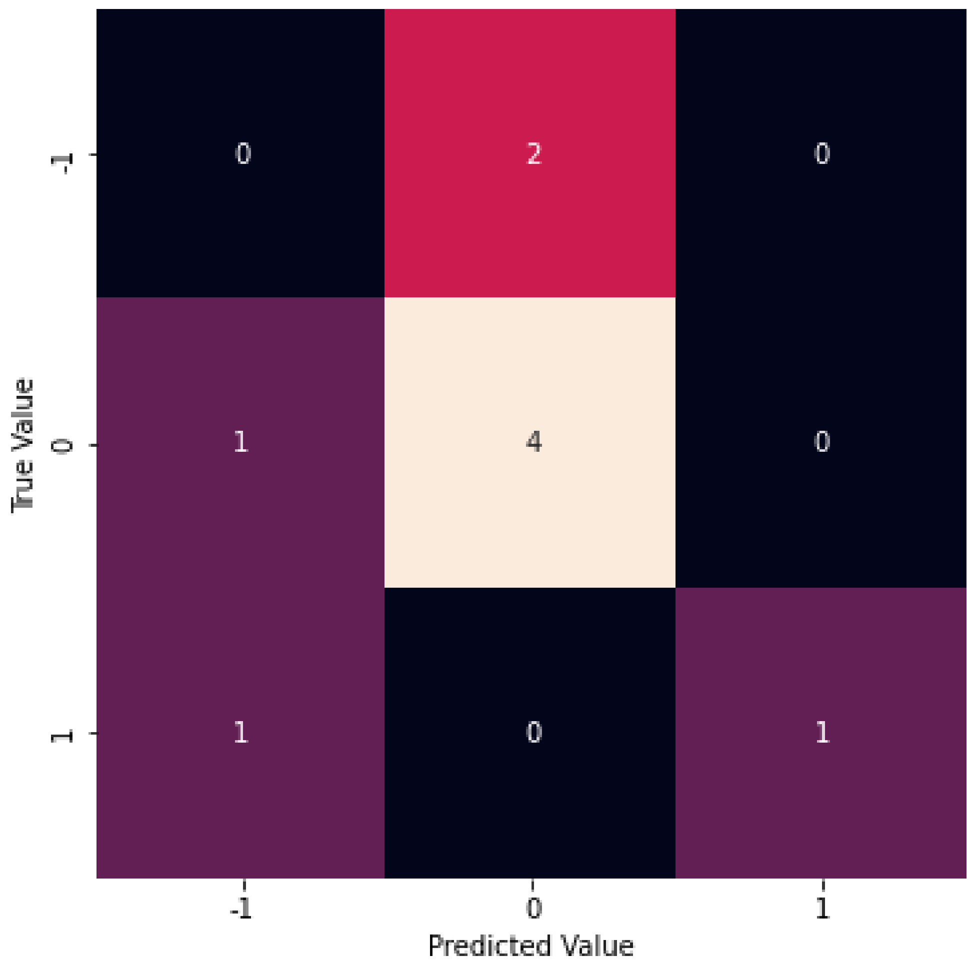
[103]:

```
temp_lst_ytest, temp_lst_yhat, temp_lst_xtest = [], [], []
for row in temp_lst:
    for i in row[4]:
        temp_lst_ytest.append(i)
    for j in row[5]:
        temp_lst_yhat.append(j)
    for k in row[6]:
        temp_lst_xtest.append(k)

cnf_ytest = np.array(temp_lst_ytest)
cnf_yhat = np.array(temp_lst_yhat)
cnf_xtest = np.array(temp_lst_xtest)
print(metrics.classification_report(cnf_ytest, cnf_yhat))
```

```
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         2
           0       0.67      0.80      0.73         5
           1       1.00      0.50      0.67         2

    accuracy                           0.56         9
   macro avg       0.56      0.43      0.46         9
weighted avg       0.59      0.56      0.55         9
```

[104]:

```
cnf_matrix = confusion_matrix(cnf_ytest, cnf_yhat)
plt.figure(figsize=(8,6))
sns.heatmap(cnf_matrix, square=True, annot=True, cbar=False)
plt.xticks([.5, 1.5, 2.5], [-1, 0, 1])
plt.yticks([.5, 1.5, 2.5], [-1, 0, 1])
plt.xlabel('Predicted Value')
plt.ylabel('True Value')
plt.show()
```

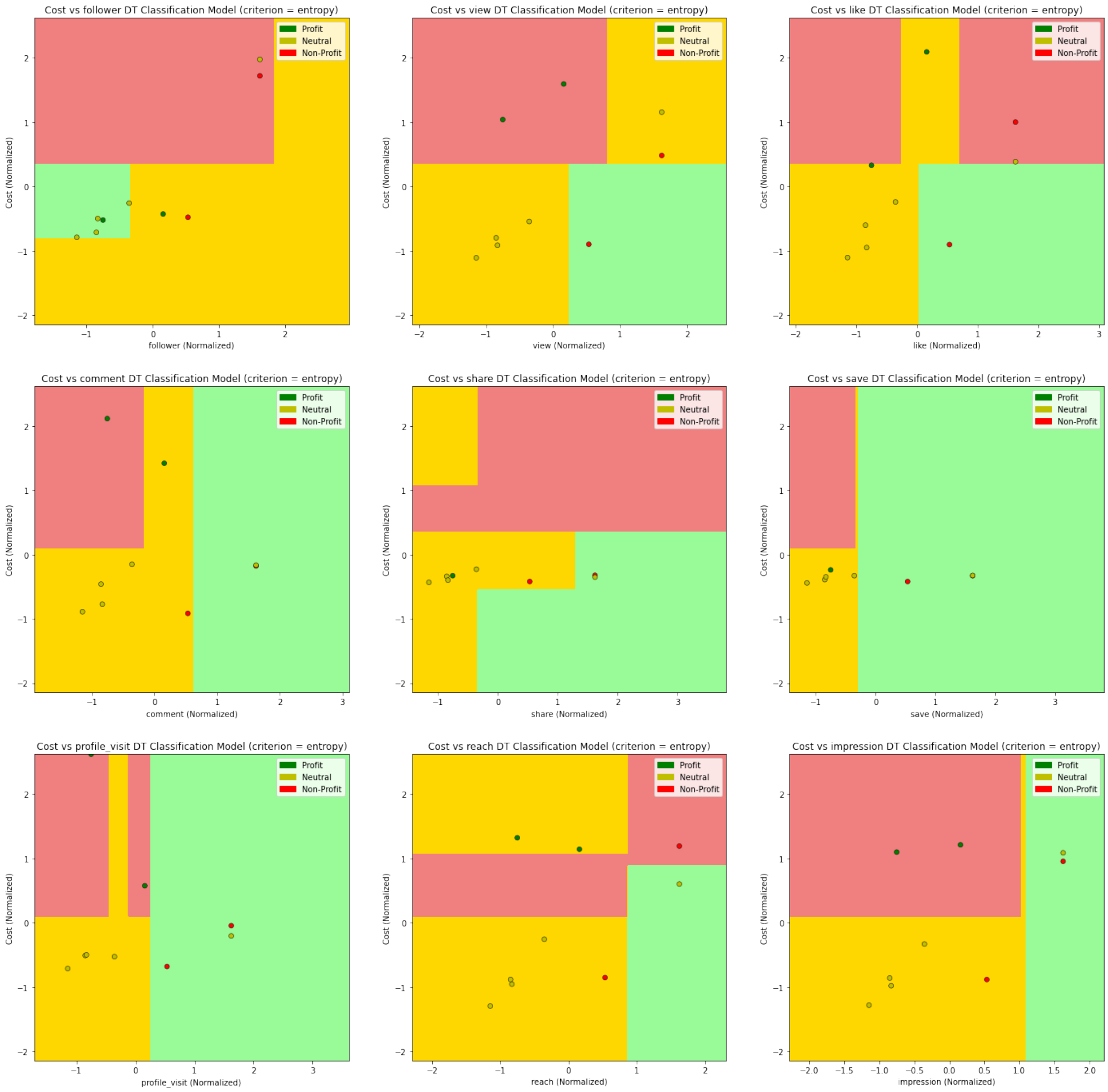


[105]:
```
fig = plt.figure(figsize = (24, 24))
ax1 = fig.add_subplot(3,3,1)
ax2 = fig.add_subplot(3,3,2)
ax3 = fig.add_subplot(3,3,3)
ax4 = fig.add_subplot(3,3,4)
ax5 = fig.add_subplot(3,3,5)
ax6 = fig.add_subplot(3,3,6)
ax7 = fig.add_subplot(3,3,7)
ax8 = fig.add_subplot(3,3,8)
ax9 = fig.add_subplot(3,3,9)



X,y = leaders_post_x[:,:10], leaders_post_y
h = .01
cmap_light = ListedColormap(['lightcoral', 'gold', 'palegreen'])
```

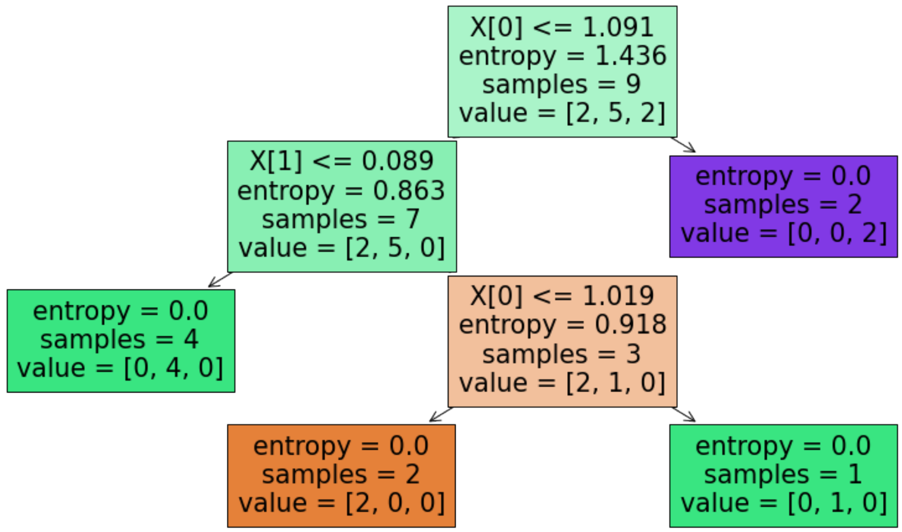
```
cmap_bold = ['r', 'y', 'g']
X_follower = X[:,(0,9)]
X_view = X[:,(1,9)]
X_like = X[:,(2,9)]
X_comment = X[:,(3,9)]
X_share = X[:,(4,9)]
X_save = X[:,(5,9)]
X_profile_visit = X[:,(6,9)]
X_reach = X[:,(7,9)]
X_impression = X[:,(8,9)]


xx_follower, xx_view, xx_like, xx_comment, xx_share, xx_save, xx_profile_visit,␣
↪xx_reach, xx_impression = None, None, None, None, None, None, None, None,␣
↪None
yy_follower, yy_view, yy_like, yy_comment, yy_share, yy_save, yy_profile_visit,␣
↪yy_reach, yy_impression = None, None, None, None, None, None, None, None,␣
↪None
Z_follower, Z_view, Z_like, Z_comment, Z_share, Z_save, Z_profile_visit,␣
↪Z_reach, Z_impression = None, None, None, None, None, None, None, None, None
g1, g2, g3, g4, g5, g6, g7, g8, g9 = None, None, None, None, None, None, None,␣
↪None, None

X_lst = [X_follower, X_view, X_like, X_comment, X_share, X_save,␣
↪X_profile_visit, X_reach, X_impression]
xx_lst = [xx_follower, xx_view, xx_like, xx_comment, xx_share, xx_save,␣
↪xx_profile_visit, xx_reach, xx_impression]
yy_lst = [yy_follower, yy_view, yy_like, yy_comment, yy_share, yy_save,␣
↪yy_profile_visit, yy_reach, yy_impression]
Z_lst = [Z_follower, Z_view, Z_like, Z_comment, Z_share, Z_save,␣
↪Z_profile_visit, Z_reach, Z_impression]
ax_lst = [ax1, ax2, ax3, ax4, ax5, ax6, ax7, ax8, ax9]
g_lst = [g1, g2, g3, g4, g5, g6, g7, g8, g9]
x_label_lst = ['follower', 'view', 'like', 'comment', 'share', 'save',␣
↪'profile_visit', 'reach', 'impression']
labels=['Profit','Neutral', 'Non-Profit']
red_patch = patches.Patch(color='r', label='Non-Profit')
yellow_patch = patches.Patch(color='y', label='Neutral')
green_patch = patches.Patch(color='g', label='Profit')


def plot_calc(x, y = y):
    '''
    This function is for calculating the area to plot with colors according to␣
↪the input
        input -> x and y.
```

```
        return -> xx, yy, Z which are needed to drawing the contour and plot.
    '''
    clf_dt.fit(x, y)
    x_min, x_max = x[:, 0].min() - 1, x[:, 0].max() + 1
    y_min, y_max = x[:, 1].min() - 1, x[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = clf_dt.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    return xx, yy, Z

for ax_, xx_, yy_, Z_, X_, x_label_, g_ in tqdm(zip(ax_lst, xx_lst, yy_lst,␣
↪Z_lst, X_lst, x_label_lst, g_lst)):
    xx_, yy_, Z_ = plot_calc(X_, y)
    ax_.contourf(xx_, yy_, Z_, cmap=cmap_light)
    g_ = sns.scatterplot(x=X_[:, 1], y=X_[:, 0], hue=leaders_post['benefit'],␣
↪palette=cmap_bold, alpha=1.0, edgecolor='black', ax=ax_)
    ax_.set_title(f'Cost vs {x_label_} DT Classification Model (criterion =␣
↪entropy)')
    ax_.set_xlim(xx_.min(), xx_.max())
    ax_.set_ylim(yy_.min(), yy_.max())
    ax_.set_xlabel(f'{x_label_} (Normalized)')
    ax_.set_ylabel('Cost (Normalized)')
    ax_.legend(handles=[green_patch, yellow_patch, red_patch],loc = 'upper␣
↪right', fontsize = 10);

plt.show()
plt.figure(figsize = (16, 8))
plot_tree(clf_dt, filled=True)
plt.show()
```

```
9it [00:01,  8.51it/s]
```



Cost vs follower DT Classification Model (criterion = entropy)
Cost vs view DT Classification Model (criterion = entropy)
Cost vs like DT Classification Model (criterion = entropy)
Cost vs comment DT Classification Model (criterion = entropy)
Cost vs share DT Classification Model (criterion = entropy)
Cost vs save DT Classification Model (criterion = entropy)
Cost vs profile_visit DT Classification Model (criterion = entropy)
Cost vs reach DT Classification Model (criterion = entropy)
Cost vs impression DT Classification Model (criterion = entropy)
Profit
Neutral
Non-Profit
Cost (Normalized)
follower (Normalized)
view (Normalized)
like (Normalized)
comment (Normalized)
share (Normalized)
save (Normalized)
profile_visit (Normalized)
reach (Normalized)
impression (Normalized)



1.2 Classification Algorithms Summary

in the table below you can see the performance summary of most accurate classification model which we tested and discussed in this notebook.

[106]:
```
data = {
    'Classification Algorithms': ['Logistic Regression', 'Support Vector␣
↪Machine', 'k-Nearest', 'Decision Tree'],
    'Advertising Post - Train Score': [0.78, 0.76, 1, 1],
    'Advertising Post - Test Score': [0.78, 0.78, 0.74, 0.72],
    'Advertising Story - Train Score': [0.96, 0.96, 1, 1],
    'Advertising Story - Test Score': [0.82, 0.82, 0.87, 0.80],
    'Influencers - Train Score': [0.92, 0.93, 1, 1],
    'Influencers - Test Score': [0.65, 0.71, 0.70, 0.78],
    'Leaders Post - Train Score': [0.59, 0.78, 0.78, 1],
    'Leaders Post - Test Score': [0.60, 0.60, 0.57, 0.56]}
score_df = pd.DataFrame(data=data)
score_df
```

[106]:
```
  Classification Algorithms  Advertising Post - Train Score  \
0       Logistic Regression                            0.78
1    Support Vector Machine                            0.76
2                 k-Nearest                            1.00
3             Decision Tree                            1.00

   Advertising Post - Test Score  Advertising Story - Train Score  \
0                           0.78                             0.96
```

```
1                          0.78                              0.96
2                          0.74                              1.00
3                          0.72                              1.00

   Advertising Story - Test Score  Influencers - Train Score  \
0                            0.82                       0.92
1                            0.82                       0.93
2                            0.87                       1.00
3                            0.80                       1.00

   Influencers - Test Score  Leaders Post - Train Score  \
0                      0.65                        0.59
1                      0.71                        0.78
2                      0.70                        0.78
3                      0.78                        1.00

   Leaders Post - Test Score
0                       0.60
1                       0.60
2                       0.57
3                       0.56
```

as you can see in the table above, beside the advertising story dataset, other datasets test accuracy are fairly mediocre, the reason behind that is the low variance and lack of data in those datasets, for instance in decision tree and k nearest algorithms, models scored perfectly in training phase but they performed about 70 ~ 80 accurate in test phase, the difference between train and test score indiciates the variance of data is not good enough for model to be abale to genaralize, even though other forms of remedy such as different architectures and regularization implemented, but they were fruitless. the only remedy and solution remaining for this problem is increasing data points in order to increasing the data variance. it's anticipated that addition of further campaings data to training phase will increase the model accuracy significantly.

---

2 Notebook by Ramin F. - @simplyramin

[ ]:

```

```