Part 7 - Decision Trees

By Aziz Presswala

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from graphviz import Source
from IPython.display import SVG
from tqdm import tqdm
from prettytable import PrettyTable

from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn import model_selection
from sklearn import metrics
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier


from gensim.models import Word2Vec
from gensim.models import KeyedVectors
```

[1.0] Splitting the Dataset into Train & Test

In [2]:
```
# Using the CleanedText column saved in final.sqlite db
con = sqlite3.connect('final.sqlite')
filtered_data = pd.read_sql_query("SELECT * FROM Reviews", con)
filtered_data.shape
```

Out[2]: (364171, 12)

In [3]:
```
# replacing all the 'positive' values of the Score attribute with 1
filtered_data['Score']=filtered_data['Score'].replace('positive',1)
```

In [4]:
```
# replacing all the 'neagtive' values of the Score attribute with 0
filtered_data['Score']=filtered_data['Score'].replace('negative',0)
```

In [5]:
```
#randomly selecting 100k points from the dataset
df=filtered_data.sample(100000)
```

In [6]:
```
#sort the dataset by timestamp
df = df.sort_values('Time')
#splitting the dataset into train(70%) & test(30%)
train_data = df[0:70000]
test_data = df[70000:100000]
```

[2.0] Featurization

[2.1] BAG OF WORDS

In [7]:
```
#applying fit transform on train datasset
count_vect = CountVectorizer(min_df=10)
x_train_bow = count_vect.fit_transform(train_data['CleanedText'].values
)
x_train_bow.shape
```

Out[7]: (70000, 7213)

Out[7]: (70000, 7213)

In [8]:
```
#applying transform on test dataset
x_test_bow = count_vect.transform(test_data['CleanedText'].values)
x_test_bow.shape
```

Out[8]: (30000, 7213)

In [9]:
```
y_train_bow = train_data['Score']
y_test_bow = test_data['Score']
```

[2.2] TF-IDF

In [22]:
```
#applying fit transform on train dataset
tf_idf_vect = TfidfVectorizer(min_df=10)
x_train_tfidf = tf_idf_vect.fit_transform(train_data['CleanedText'].val
ues)
x_train_tfidf.shape
```

Out[22]: (70000, 7213)

In [23]:
```
#applying transform on test dataset
x_test_tfidf = tf_idf_vect.transform(test_data['CleanedText'].values)
x_test_tfidf.shape
```

Out[23]: (30000, 7213)

In [24]:
```
y_train_tfidf = train_data['Score']
y_test_tfidf = test_data['Score']
```

[2.3] Avg. Word2Vec

In [34]:
```
#training Word2Vec Model for train dataset
i=0
list_of_sent=[]
```

```
for sent in train_data['CleanedText'].values:
    list_of_sent.append(sent.split())
```

In [35]:
```
w2v_model=Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
```

In [36]:
```
X = w2v_model[w2v_model.wv.vocab]
```

In [37]:
```
#computing Avg Word2Vec for train dataset
w2v_words = list(w2v_model.wv.vocab)
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sent): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|██████████████████████████████████████████████████████████████████
█████████| 70000/70000 [03:09<00:00, 368.44it/s]
```

```
70000
50
```

In [38]:
```
x_train_w2v = np.array(sent_vectors)
y_train_w2v = train_data['Score']
x_train_w2v.shape
```

Out[38]:
```
(70000, 50)
```

In [39]:
```
#training Word2Vec Model for test dataset
i=0
list_of_sent1=[]
for sent in test_data['CleanedText'].values:
    list_of_sent1.append(sent.split())
```

In [40]:
```
#computing Avg Word2Vec for test dataset
w2v_words = list(w2v_model.wv.vocab)
sent_vectors = []; # the avg-w2v for each sentence/review is stored in
 this list
for sent in tqdm(list_of_sent1): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████████
█████████| 30000/30000 [01:18<00:00, 382.73it/s]
```

```
30000
50
```

In [41]:
```
x_test_w2v = np.array(sent_vectors)
y_test_w2v = test_data['Score']
x_test_w2v.shape
```

Out[41]:
```
(30000, 50)
```

[2.4] TFIDF - Word2Vec

In [49]:
```
# training model for training data
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(train_data['CleanedText'].values)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [50]:
```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and ce
ll_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is st
ored in this list
row=0;
for sent in tqdm(list_of_sent): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/r
eview
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            # tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████████████████████████
████████| 70000/70000 [03:51<00:00, 302.89it/s]
```

In [51]:
```
x_train_tfw2v = np.array(tfidf_sent_vectors)
y_train_tfw2v = train_data['Score']
x_train_tfw2v.shape
```

Out[51]: (70000, 50)

In [52]:
```
# training model for test dataset
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(test_data['CleanedText'].values)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [53]:
```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and ce
ll_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is st
ored in this list
row=0;
for sent in tqdm(list_of_sent1): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/r
eview
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            # tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████████████████
█████████| 30000/30000 [01:09<00:00, 433.40it/s]
```

In [54]:
```python
x_test_tfw2v = np.array(tfidf_sent_vectors)
y_test_tfw2v = test_data['Score']
x_test_tfw2v.shape
```

Out[54]:
```
(30000, 50)
```

[3.0] Applying Decision Trees

[3.1] Applying Decision Trees on BOW, SET 1

In [10]:
```python
# initializing DecisionTreeClassifier model
dtc = DecisionTreeClassifier()

# hyperparameter values we need to try on classifier
max_depth = [1, 10, 25, 50, 100, 500]
min_samples_split  = [5, 10, 25, 50, 100, 500]
param_grid = {'max_depth':[1, 10, 25, 50, 100, 500],
              'min_samples_split':[5, 10, 25, 50, 100, 500]}

# using GridSearchCV to find the optimal value of hyperparameters
# using roc_auc as the scoring parameter & applying 5 fold CV
gscv = GridSearchCV(dtc,param_grid,scoring='roc_auc',cv=5,n_jobs=-1,ret
urn_train_score=True)

gscv.fit(x_train_bow,y_train_bow)
print("Best Max Depth Value:",gscv.best_params_['max_depth'])
print("Best Min Sample Split Value:",gscv.best_params_['min_samples_spl
it'])
print("Best ROC AUC Score: %.5f"%(gscv.best_score_))
```

```
Best Max Depth Value: 50
Best Min Sample Split Value: 500
Best ROC AUC Score: 0.82853
```

In [11]:

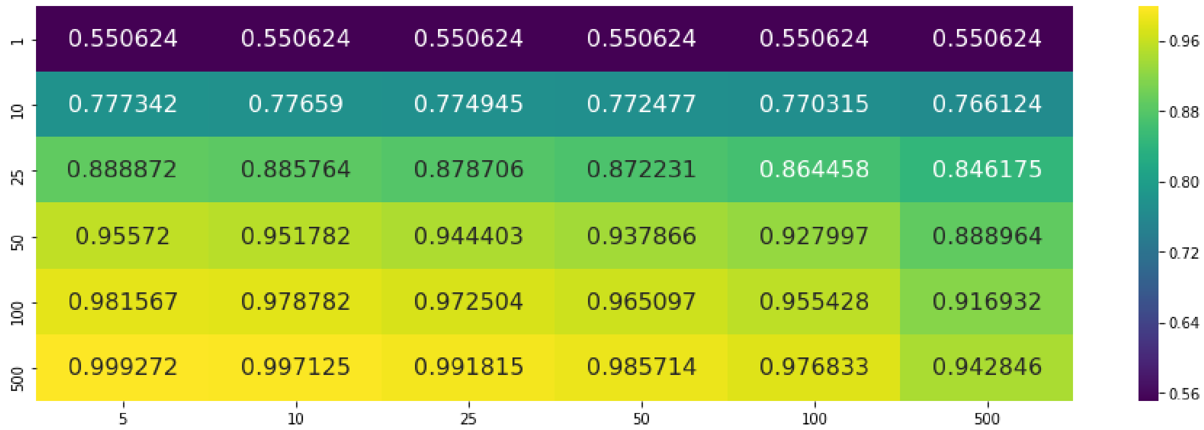
```
# determining optimal depth and sample split values
optimal_depth = gscv.best_params_['max_depth']
optimal_sample_split = gscv.best_params_['min_samples_split']

#training the model using the optimal hyperparameters
dtc_clf = DecisionTreeClassifier(max_depth=optimal_depth, min_samples_s
plit=optimal_sample_split)
dtc_clf.fit(x_train_bow,y_train_bow)

#predicting the class label using test data
y_pred = dtc_clf.predict_proba(x_test_bow)[:,1]

#determining the Test roc_auc_score for optimal hyperparameters
auc_score = roc_auc_score(y_test_bow, y_pred)
print('\n**** Test roc_auc_score is %f ****' % (auc_score))
```

```
**** Test roc_auc_score is 0.831107 ****
```

Seaborn Heatmap on Train Data

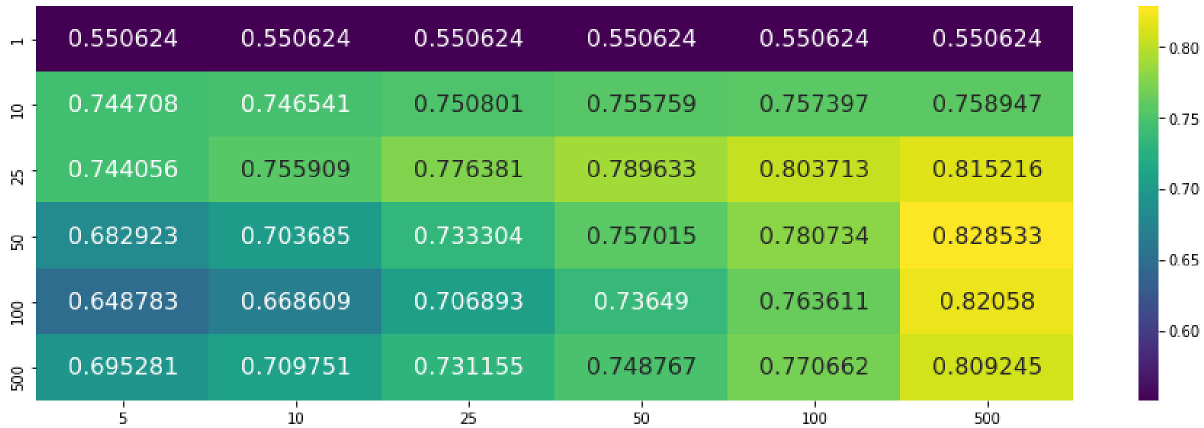
In [12]:

```
A=np.array(gscv.cv_results_['mean_train_score'])
B = np.reshape(A, (6,6))
df = pd.DataFrame(B, index=max_depth, columns=min_samples_split)
plt.figure(figsize = (16,5))
sns.heatmap(df, annot=True, annot_kws={"size": 16}, fmt="g", cmap='viri
dis')
plt.show()
```



Seaborn Heatmap on Test Data

In [13]:
```
A=np.array(gscv.cv_results_['mean_test_score'])
B = np.reshape(A, (6,6))
df = pd.DataFrame(B, index=max_depth, columns=min_samples_split)
plt.figure(figsize = (16,5))
sns.heatmap(df, annot=True, annot_kws={"size": 16}, fmt="g", cmap='viri
dis')
plt.show()
```

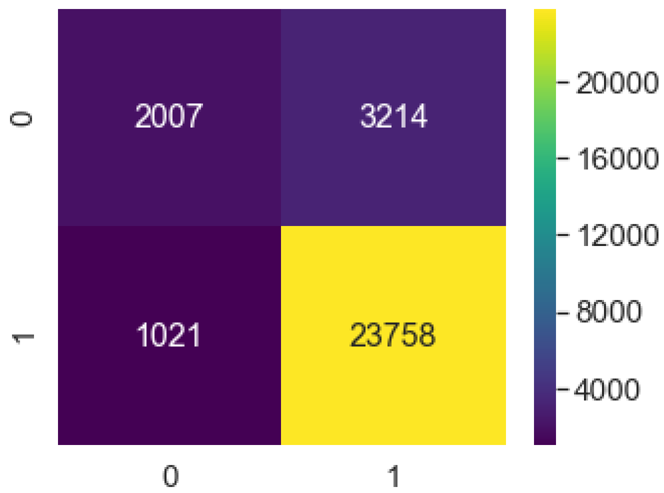


Confusion Matrix on Test Data

In [14]:
```
# plotting confusion matrix as heatmap
y_predict = dtc_clf.predict(x_test_bow)
cm = confusion_matrix(y_test_bow, y_predict)
print(cm)
plt.figure(figsize = (5,4))
df_cm = pd.DataFrame(cm, range(2),range(2))
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g',cmap='vir
idis')
```

```
[[ 2007  3214]
 [ 1021 23758]]
```

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x1ee655c5da0>



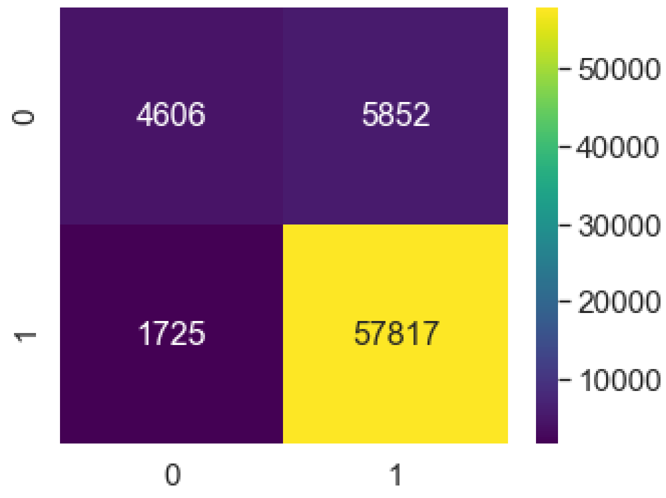
Confusion Matrix on Train Data

In [15]:
```
# plotting confusion matrix as heatmap
y_predict = dtc_clf.predict(x_train_bow)
cm = confusion_matrix(y_train_bow, y_predict)
```

```
print(cm)
plt.figure(figsize = (5,4))
df_cm = pd.DataFrame(cm, range(2),range(2))
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g',cmap='vir
idis')
```

```
[[ 4606  5852]
 [ 1725 57817]]
```

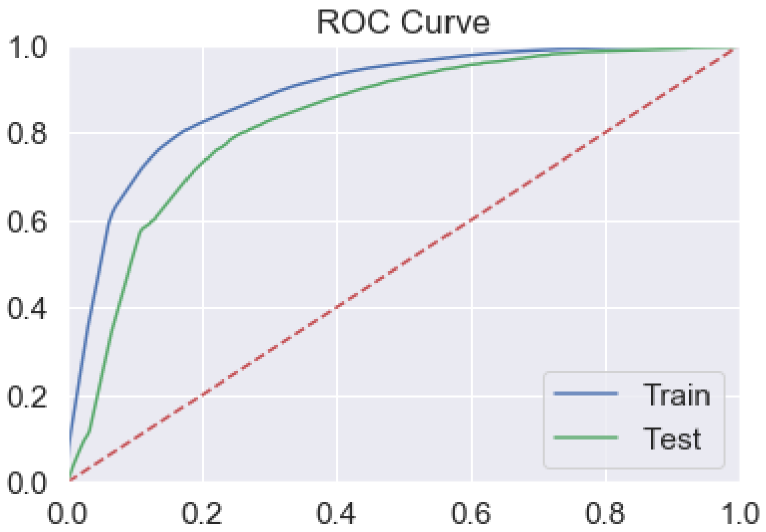
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x1ee685c4860>



ROC Curve

In [16]:
```
# Plotting roc curve on Train Data
pred_train = dtc_clf.predict_proba(x_train_bow)[:,1]
fpr, tpr, threshold = roc_curve(y_train_bow, pred_train)
plt.plot(fpr, tpr, 'b', label='Train')

# Plotting roc curve on Test Data
pred_test = dtc_clf.predict_proba(x_test_bow)[:,1]
fpr, tpr, threshold = roc_curve(y_test_bow, pred_test)
plt.plot(fpr, tpr, 'g', label='Test')
```

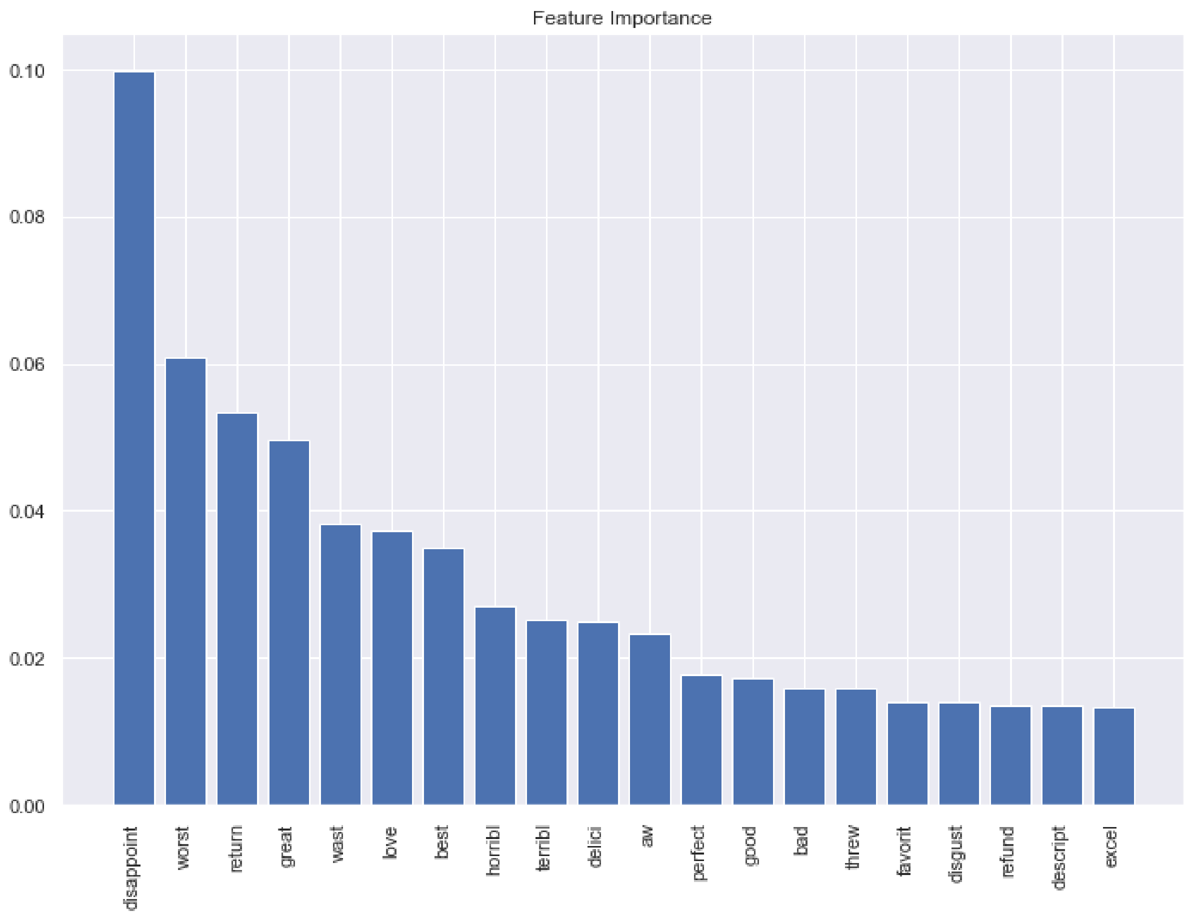
```
plt.title('ROC Curve')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```



Top 20 important features from SET 1

In [17]:
```
# Calculate feature importances from decision trees
importances = dtc_clf.feature_importances_

# Sort feature importances in descending order and get their indices
indices = np.argsort(importances)[::-1][:20]

# Get the feature names from the vectorizer
names = count_vect.get_feature_names()

sns.set(rc={'figure.figsize':(11.7,8.27)})
```

```python
# Create plot
plt.figure()

# Create plot title
plt.title("Feature Importance")

# Add bars
plt.bar(range(20), importances[indices])

# Add feature names as x-axis labels
names = np.array(names)
plt.xticks(range(20), names[indices], rotation=90)

# Show plot
plt.show()
```



Graphviz visualization of Decision Tree on BOW, SET 1

In [21]:
```
graph = Source( tree.export_graphviz(dtc_clf, out_file=None, class_name
s=['negative','positive'],
                                     filled=True, rounded=True, feature
_names=names))
SVG(graph.pipe(format='svg'))
graph.view()
# graphviz graph uploaded as pdf
```

Out[21]:

```
'Source.gv.pdf'
```

[3.2] Applying Decision Trees on TFIDF, SET 2

In [25]:

```
# initializing DecisionTreeClassifier model
dtc = DecisionTreeClassifier()

# hyperparameter values we need to try on classifier
max_depth = [1, 10, 25, 50, 100, 500]
min_samples_split  = [5, 10, 25, 50, 100, 500]
param_grid = {'max_depth':[1, 10, 25, 50, 100, 500],
              'min_samples_split':[5, 10, 25, 50, 100, 500]}

# using GridSearchCV to find the optimal value of hyperparameters
# using roc_auc as the scoring parameter & applying 5 fold CV
gscv = GridSearchCV(dtc,param_grid,scoring='roc_auc',cv=5,n_jobs=-1,ret
urn_train_score=True)

gscv.fit(x_train_tfidf,y_train_tfidf)
print("Best Max Depth Value:",gscv.best_params_['max_depth'])
print("Best Min Sample Split Value:",gscv.best_params_['min_samples_spl
it'])
print("Best ROC AUC Score: %.5f"%(gscv.best_score_))
```

```
Best Max Depth Value: 50
Best Min Sample Split Value: 500
Best ROC AUC Score: 0.81669
```

In [26]:

```
# determining optimal depth and sample split values
optimal_depth = gscv.best_params_['max_depth']
optimal_sample_split = gscv.best_params_['min_samples_split']

#training the model using the optimal hyperparameters
dtc_clf = DecisionTreeClassifier(max_depth=optimal_depth, min_samples_s
plit=optimal_sample_split)
dtc_clf.fit(x_train_tfidf,y_train_tfidf)
```

```
#predicting the class label using test data
y_pred = dtc_clf.predict_proba(x_test_tfidf)[:,1]

#determining the Test roc_auc_score for optimal hyperparameters
auc_score = roc_auc_score(y_test_tfidf, y_pred)
print('\n**** Test roc_auc_score is %f ****' % (auc_score))
```

```
**** Test roc_auc_score is 0.821965 ****
```

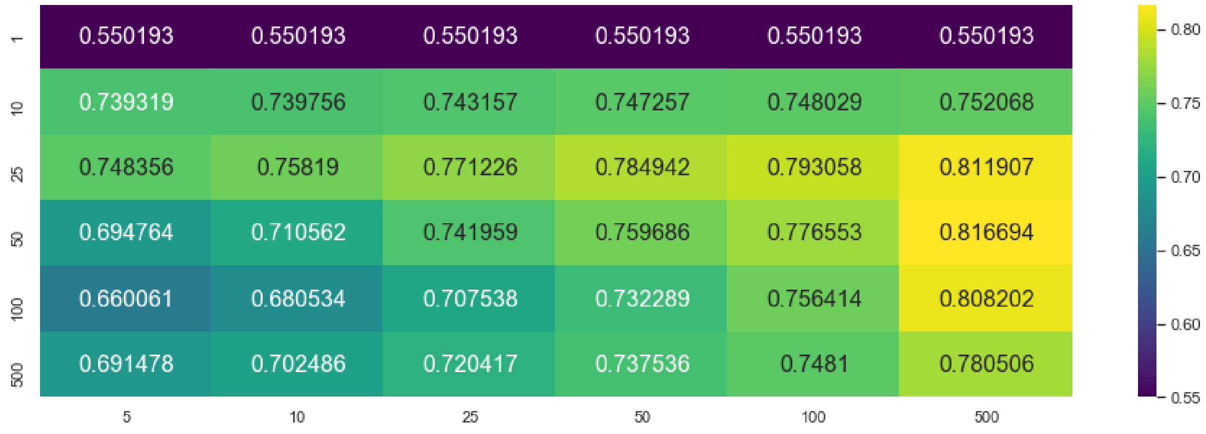
Seaborn Heatmap on Train Data

In [27]:
```
A=np.array(gscv.cv_results_['mean_train_score'])
B = np.reshape(A, (6,6))
df = pd.DataFrame(B, index=max_depth, columns=min_samples_split)
plt.figure(figsize = (16,5))
sns.heatmap(df, annot=True, annot_kws={"size": 16}, fmt="g", cmap='viri
dis')
plt.show()
```

Seaborn Heatmap on Test Data

In [28]:
```
A=np.array(gscv.cv_results_['mean_test_score'])
```

```
B = np.reshape(A, (6,6))
df = pd.DataFrame(B, index=max_depth, columns=min_samples_split)
plt.figure(figsize = (16,5))
sns.heatmap(df, annot=True, annot_kws={"size": 16}, fmt="g", cmap='viri
dis')
plt.show()
```

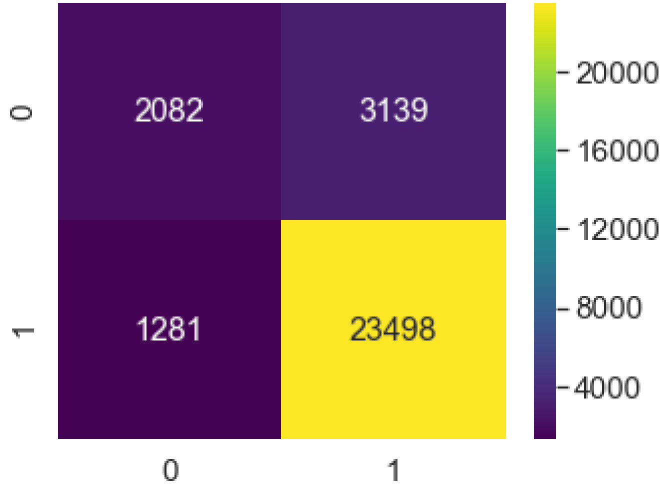


Confusion Matrix on Test Data

In [29]:
```
# plotting confusion matrix as heatmap
y_predict = dtc_clf.predict(x_test_tfidf)
cm = confusion_matrix(y_test_tfidf, y_predict)
print(cm)
plt.figure(figsize = (5,4))
df_cm = pd.DataFrame(cm, range(2),range(2))
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g',cmap='vir
idis')
```

```
[[ 2082  3139]
 [ 1281 23498]]
```

Out[29]: `<matplotlib.axes._subplots.AxesSubplot at 0x1ee02dd6d30>`

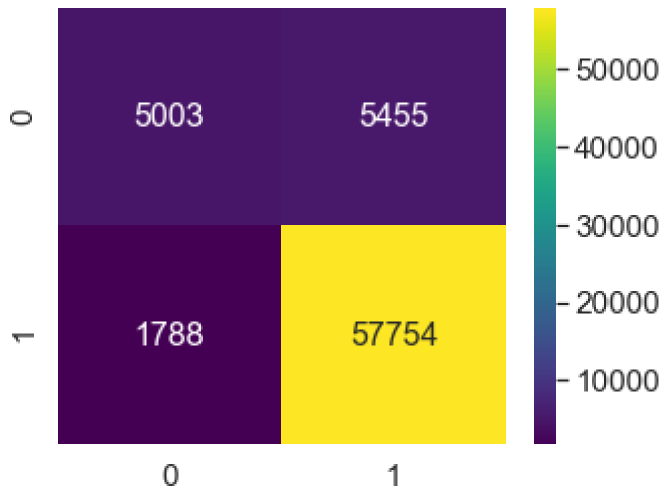


Confusion Matrix on Train Data

In [30]:
```
# plotting confusion matrix as heatmap
y_predict = dtc_clf.predict(x_train_tfidf)
cm = confusion_matrix(y_train_tfidf, y_predict)
print(cm)
plt.figure(figsize = (5,4))
df_cm = pd.DataFrame(cm, range(2),range(2))
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g',cmap='vir
idis')
```

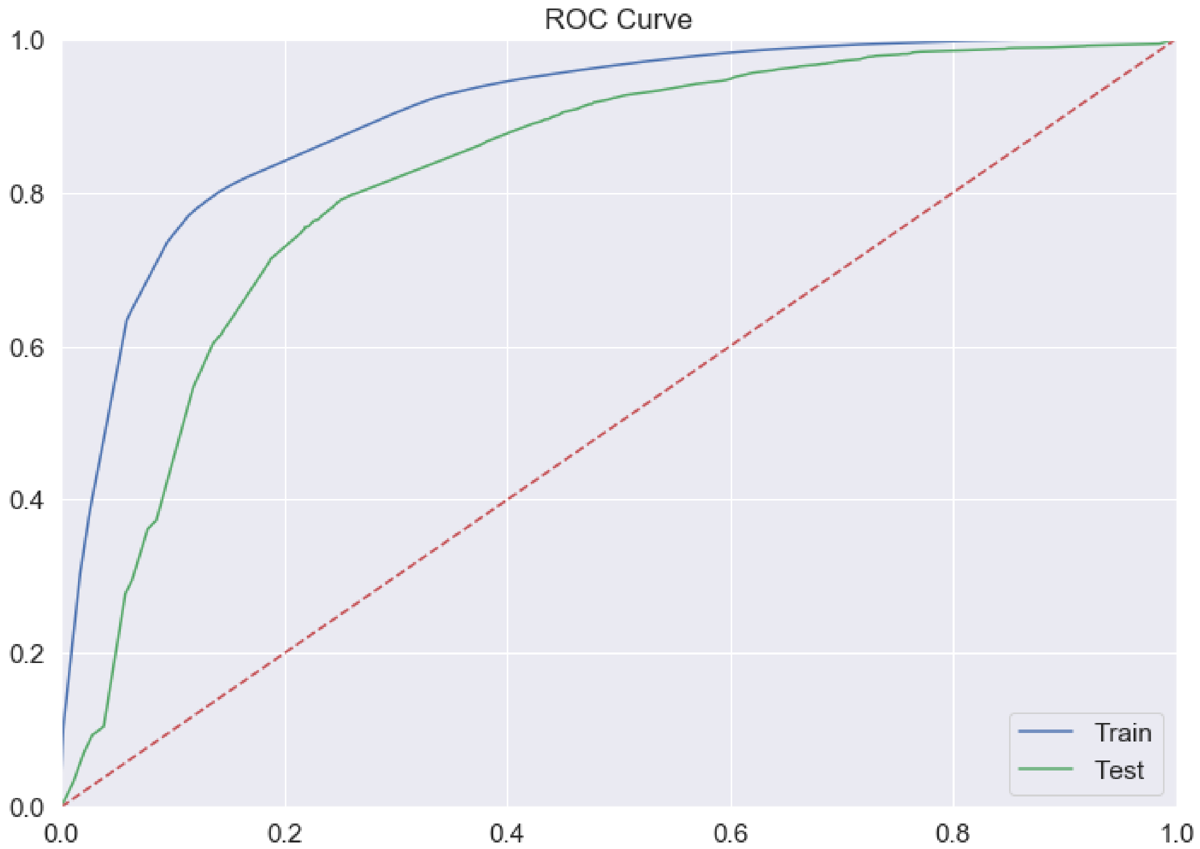
```
[[ 5003  5455]
 [ 1788 57754]]
```

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x1ee7f719940>



ROC Curve

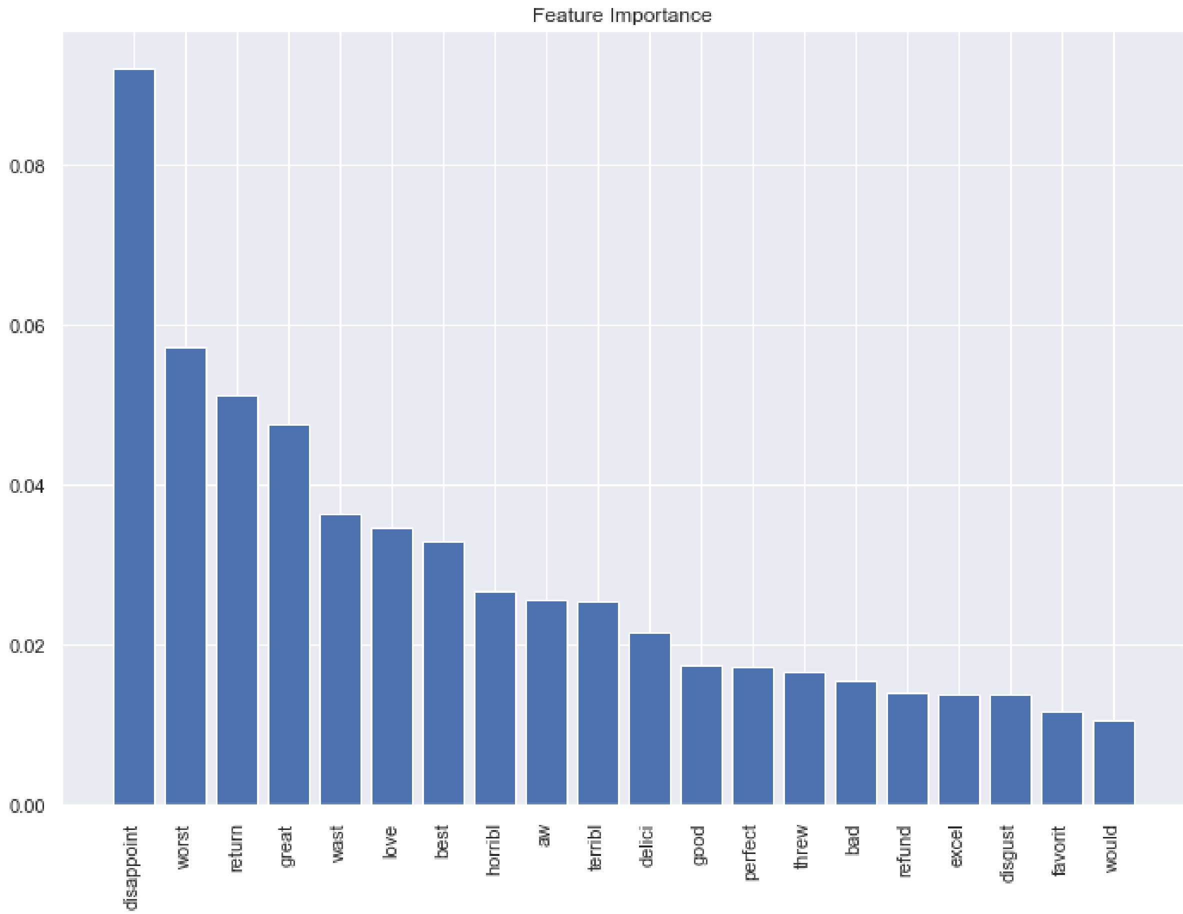
In [31]:
```
# Plotting roc curve on Train Data
pred_train = dtc_clf.predict_proba(x_train_tfidf)[:,1]
fpr, tpr, threshold = roc_curve(y_train_tfidf, pred_train)
plt.plot(fpr, tpr, 'b', label='Train')

# Plotting roc curve on Test Data
pred_test = dtc_clf.predict_proba(x_test_tfidf)[:,1]
fpr, tpr, threshold = roc_curve(y_test_tfidf, pred_test)
plt.plot(fpr, tpr, 'g', label='Test')

plt.title('ROC Curve')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```



Top 20 important features from SET 2

In [32]:
```
# Calculate feature importances from decision trees
importances = dtc_clf.feature_importances_

# Sort feature importances in descending order and get their indices
indices = np.argsort(importances)[::-1][:20]

# Get the feature names from the vectorizer
names = tf_idf_vect.get_feature_names()
```

```python
sns.set(rc={'figure.figsize':(11.7,8.27)})

# Create plot
plt.figure()

# Create plot title
plt.title("Feature Importance")

# Add bars
plt.bar(range(20), importances[indices])

# Add feature names as x-axis labels
names = np.array(names)
plt.xticks(range(20), names[indices], rotation=90)

# Show plot
plt.show()
```



Graphviz visualization of Decision Tree on TFIDF, SET 2

In [33]:
```
graph = Source( tree.export_graphviz(dtc_clf, out_file=None, class_names=['negative','positive'],
                                                       filled=True, rounded=True, feature_names=names))
SVG(graph.pipe(format='svg'))
graph.view()
# graphviz graph uploaded as pdf
```

Out[33]: 'Source.gv.pdf'

[3.3] Applying Decision Trees on AVG W2V, SET 3

In [42]:
```
# initializing DecisionTreeClassifier model
dtc = DecisionTreeClassifier()

# hyperparameter values we need to try on classifier
max_depth = [1, 10, 25, 50, 100, 500]
min_samples_split  = [10, 25, 50, 100, 500, 1000]
param_grid = {'max_depth':[1, 10, 25, 50, 100, 500],
              'min_samples_split':[10, 25, 50, 100, 500, 1000]}

# using GridSearchCV to find the optimal value of hyperparameters
# using roc_auc as the scoring parameter & applying 5 fold CV
gscv = GridSearchCV(dtc,param_grid,scoring='roc_auc',cv=5,n_jobs=-1,ret
urn_train_score=True)

gscv.fit(x_train_w2v,y_train_w2v)
print("Best Max Depth Value:",gscv.best_params_['max_depth'])
print("Best Min Sample Split Value:",gscv.best_params_['min_samples_spl
it'])
print("Best ROC AUC Score: %.5f"%(gscv.best_score_))
```

```
Best Max Depth Value: 10
Best Min Sample Split Value: 500
Best ROC AUC Score: 0.81900
```

In [43]:
```
# determining optimal depth and sample split values
optimal_depth = gscv.best_params_['max_depth']
optimal_sample_split = gscv.best_params_['min_samples_split']

#training the model using the optimal hyperparameters
dtc_clf = DecisionTreeClassifier(max_depth=optimal_depth, min_samples_s
plit=optimal_sample_split)
dtc_clf.fit(x_train_w2v,y_train_w2v)
```

```
#predicting the class label using test data
y_pred = dtc_clf.predict_proba(x_test_w2v)[:,1]

#determining the Test roc_auc_score for optimal hyperparameters
auc_score = roc_auc_score(y_test_w2v, y_pred)
print('\n**** Test roc_auc_score is %f ****' % (auc_score))
```

```
**** Test roc_auc_score is 0.823780 ****
```

Seaborn Heatmap on Train Data

In [44]:
```
A=np.array(gscv.cv_results_['mean_train_score'])
B = np.reshape(A, (6,6))
df = pd.DataFrame(B, index=max_depth, columns=min_samples_split)
plt.figure(figsize = (16,5))
sns.heatmap(df, annot=True, annot_kws={"size": 16}, fmt="g", cmap='viri
dis')
plt.show()
```

Seaborn Heatmap on Test Data

In [45]:
```
A=np.array(gscv.cv_results_['mean_test_score'])
```

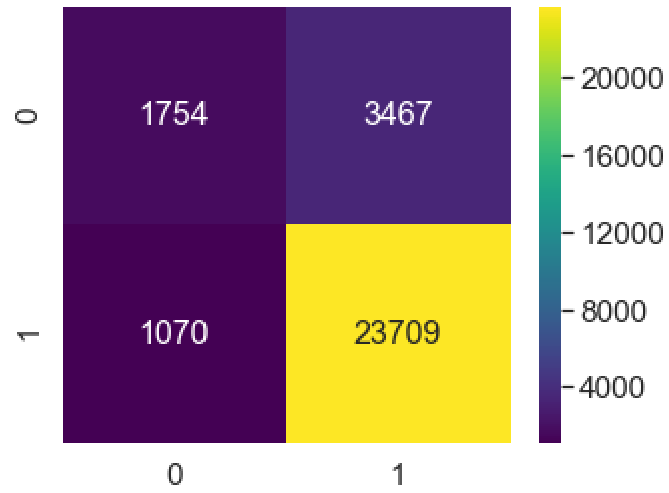
```
B = np.reshape(A, (6,6))
df = pd.DataFrame(B, index=max_depth, columns=min_samples_split)
plt.figure(figsize = (16,5))
sns.heatmap(df, annot=True, annot_kws={"size": 16}, fmt="g", cmap='viri
dis')
plt.show()
```

Confusion Matrix on Test Data

In [46]:
```
# plotting confusion matrix as heatmap
y_predict = dtc_clf.predict(x_test_w2v)
cm = confusion_matrix(y_test_w2v, y_predict)
print(cm)
plt.figure(figsize = (5,4))
df_cm = pd.DataFrame(cm, range(2),range(2))
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g',cmap='vir
idis')
```

```
[[ 1754  3467]
 [ 1070 23709]]
```

Out[46]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1ee176658d0>
```

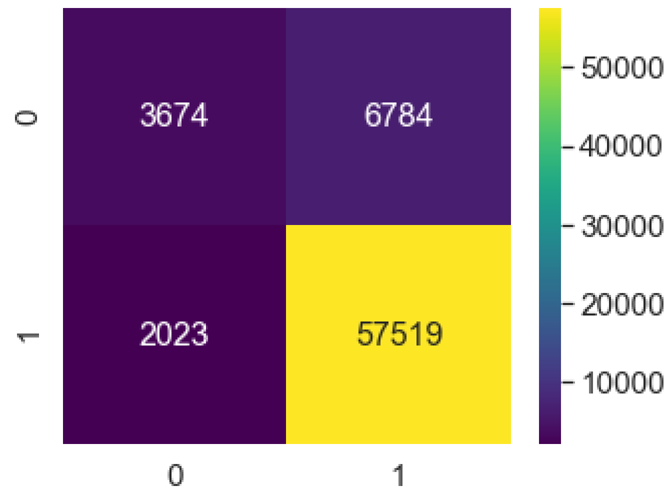


Confusion Matrix on Train Data

In [47]:
```
# plotting confusion matrix as heatmap
y_predict = dtc_clf.predict(x_train_w2v)
cm = confusion_matrix(y_train_w2v, y_predict)
print(cm)
plt.figure(figsize = (5,4))
df_cm = pd.DataFrame(cm, range(2),range(2))
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g',cmap='viridis')
```

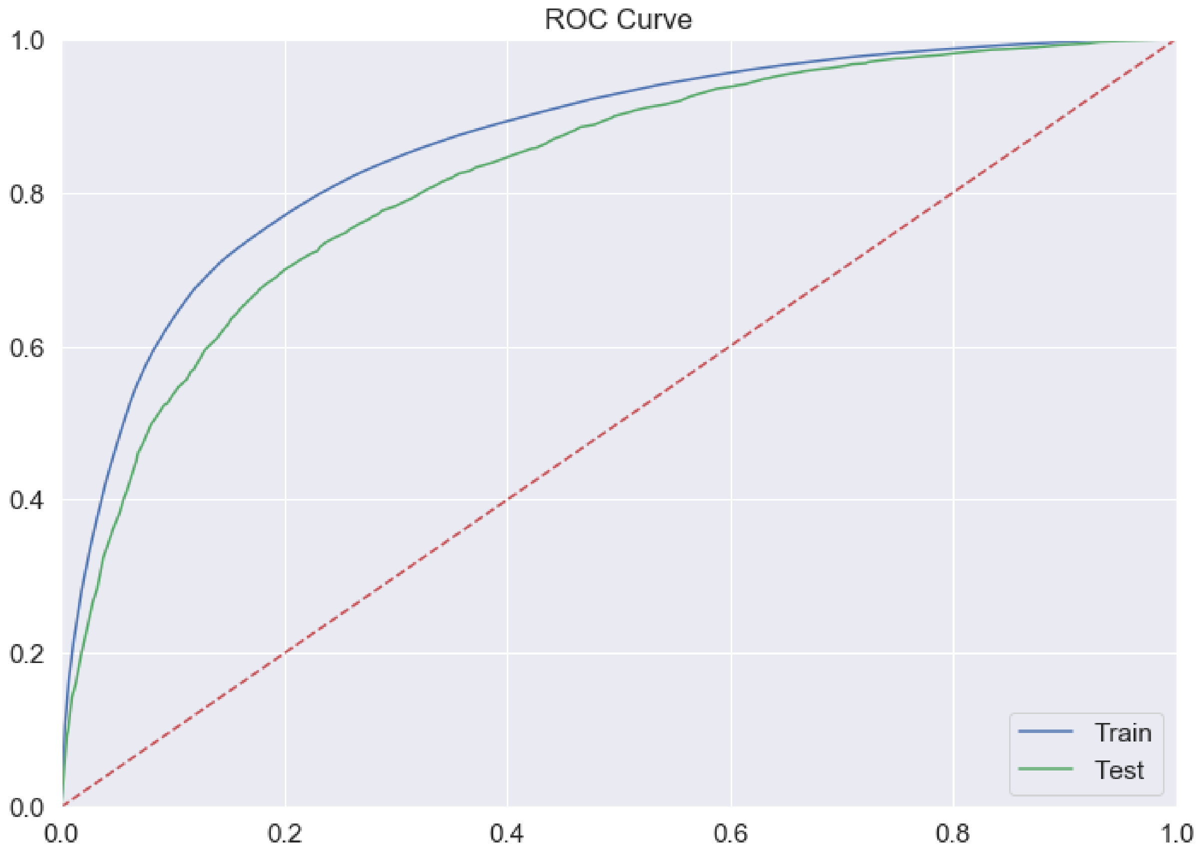
```
[[ 3674  6784]
 [ 2023 57519]]
```

Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0x1ee176bda20>



ROC Curve

In [48]:
```
# Plotting roc curve on Train Data
pred_train = dtc_clf.predict_proba(x_train_w2v)[:,1]
fpr, tpr, threshold = roc_curve(y_train_w2v, pred_train)
plt.plot(fpr, tpr, 'b', label='Train')

# Plotting roc curve on Test Data
pred_test = dtc_clf.predict_proba(x_test_w2v)[:,1]
fpr, tpr, threshold = roc_curve(y_test_w2v, pred_test)
plt.plot(fpr, tpr, 'g', label='Test')

plt.title('ROC Curve')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```



[3.4] Applying Decision Trees on TFIDF W2V, SET 4

In [55]:
```
# initializing DecisionTreeClassifier model
dtc = DecisionTreeClassifier()

# hyperparameter values we need to try on classifier
max_depth = [1, 10, 25, 50, 100, 500]
min_samples_split  = [10, 25, 50, 100, 500, 1000]
param_grid = {'max_depth':[1, 10, 25, 50, 100, 500],
              'min_samples_split':[10, 25, 50, 100, 500, 1000]}
```

```
# using GridSearchCV to find the optimal value of hyperparameters
# using roc_auc as the scoring parameter & applying 5 fold CV
gscv = GridSearchCV(dtc,param_grid,scoring='roc_auc',cv=5,n_jobs=-1,ret
urn_train_score=True)

gscv.fit(x_train_tfw2v,y_train_tfw2v)
print("Best Max Depth Value:",gscv.best_params_['max_depth'])
print("Best Min Sample Split Value:",gscv.best_params_['min_samples_spl
it'])
print("Best ROC AUC Score: %.5f"%(gscv.best_score_))
```

```
Best Max Depth Value: 10
Best Min Sample Split Value: 500
Best ROC AUC Score: 0.78579
```

In [56]:

```
# determining optimal depth and sample split values
optimal_depth = gscv.best_params_['max_depth']
optimal_sample_split = gscv.best_params_['min_samples_split']

#training the model using the optimal hyperparameters
dtc_clf = DecisionTreeClassifier(max_depth=optimal_depth, min_samples_s
plit=optimal_sample_split)
dtc_clf.fit(x_train_tfw2v,y_train_tfw2v)

#predicting the class label using test data
y_pred = dtc_clf.predict_proba(x_test_tfw2v)[:,1]

#determining the Test roc_auc_score for optimal hyperparameters
auc_score = roc_auc_score(y_test_tfw2v, y_pred)
print('\n**** Test roc_auc_score is %f ****' % (auc_score))
```

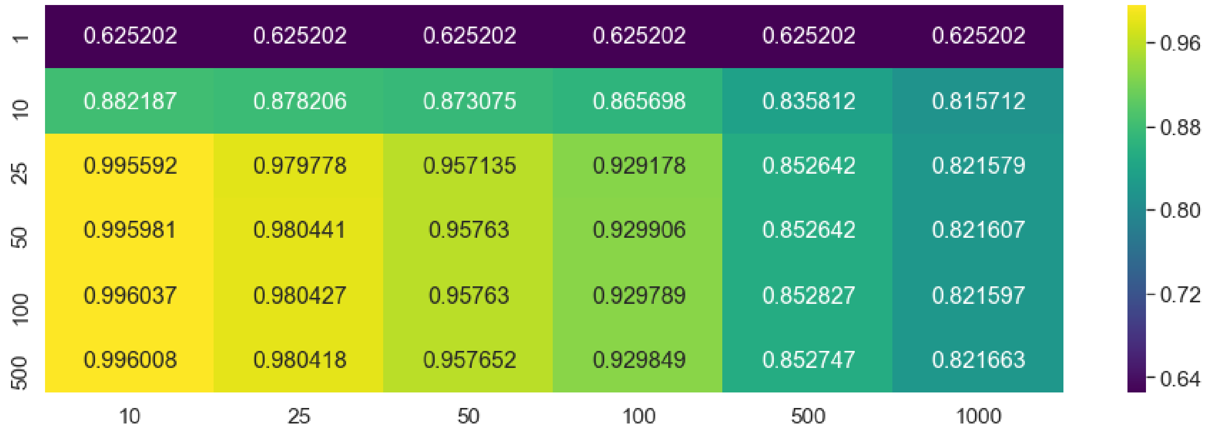
```
**** Test roc_auc_score is 0.784230 ****
```

Seaborn Heatmap on Train Data

In [57]:

```
A=np.array(gscv.cv_results_['mean_train_score'])
B = np.reshape(A, (6,6))
df = pd.DataFrame(B, index=max_depth, columns=min_samples_split)
```

```
plt.figure(figsize = (16,5))
sns.heatmap(df, annot=True, annot_kws={"size": 16}, fmt="g", cmap='viri
dis')
plt.show()
```



Seaborn Heatmap on Test Data

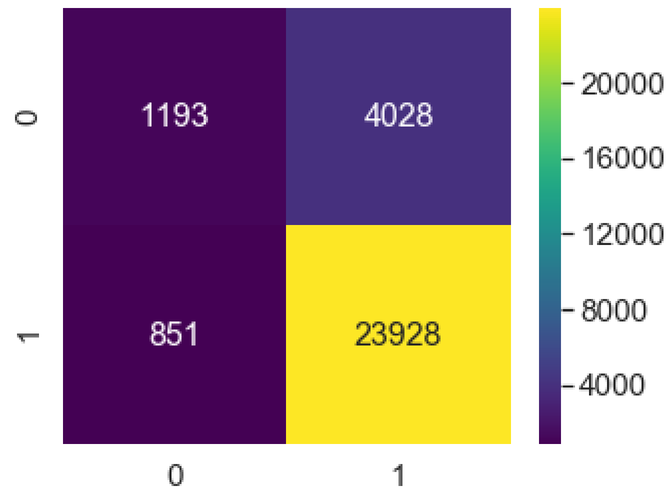
In [58]:
```
A=np.array(gscv.cv_results_['mean_test_score'])
B = np.reshape(A, (6,6))
df = pd.DataFrame(B, index=max_depth, columns=min_samples_split)
plt.figure(figsize = (16,5))
sns.heatmap(df, annot=True, annot_kws={"size": 16}, fmt="g", cmap='viri
dis')
plt.show()
```

Confusion Matrix on Test Data

In [59]:
```
# plotting confusion matrix as heatmap
y_predict = dtc_clf.predict(x_test_tfw2v)
cm = confusion_matrix(y_test_tfw2v, y_predict)
print(cm)
plt.figure(figsize = (5,4))
df_cm = pd.DataFrame(cm, range(2),range(2))
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g',cmap='vir
idis')
```

```
[[ 1193  4028]
 [  851 23928]]
```

Out[59]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1ee17602208>
```

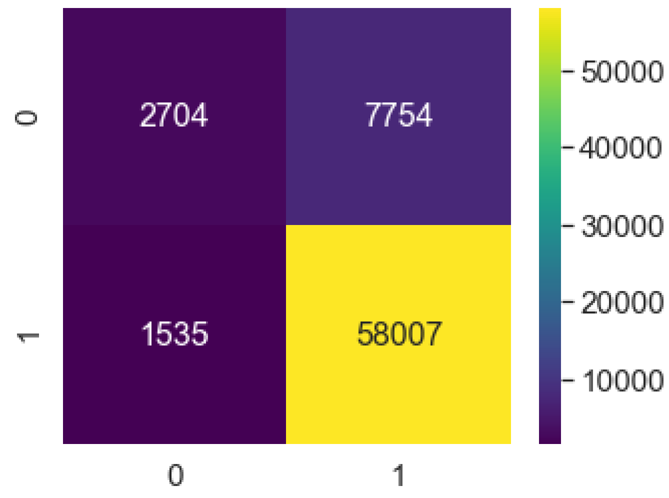


Confusion Matrix on Train Data

In [60]:
```
# plotting confusion matrix as heatmap
y_predict = dtc_clf.predict(x_train_tfw2v)
cm = confusion_matrix(y_train_tfw2v, y_predict)
print(cm)
plt.figure(figsize = (5,4))
df_cm = pd.DataFrame(cm, range(2),range(2))
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g',cmap='vir
idis')
```

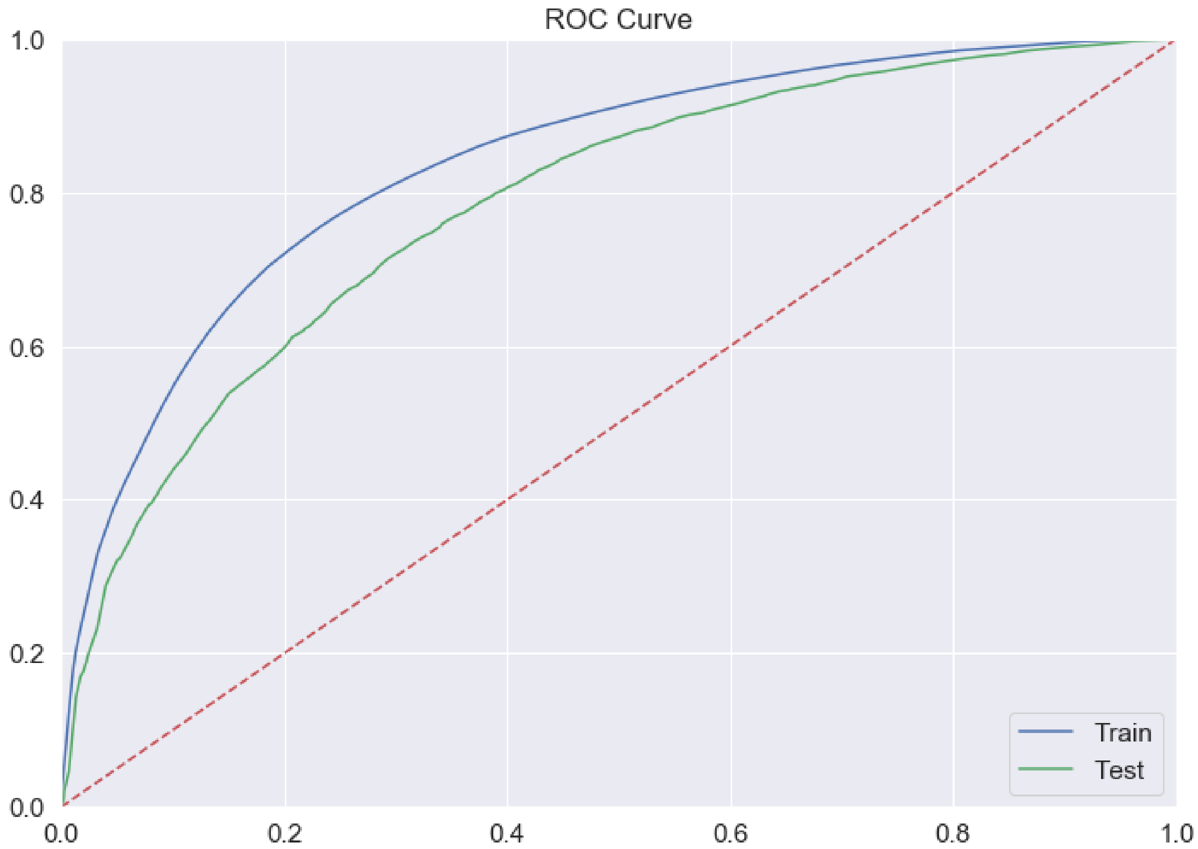
```
[[ 2704  7754]
 [ 1535 58007]]
```

Out[60]: <matplotlib.axes._subplots.AxesSubplot at 0x1ee188e7668>



ROC Curve

In [61]:
```
# Plotting roc curve on Train Data
pred_train = dtc_clf.predict_proba(x_train_tfw2v)[:,1]
fpr, tpr, threshold = roc_curve(y_train_tfw2v, pred_train)
plt.plot(fpr, tpr, 'b', label='Train')

# Plotting roc curve on Test Data
pred_test = dtc_clf.predict_proba(x_test_tfw2v)[:,1]
fpr, tpr, threshold = roc_curve(y_test_tfw2v, pred_test)
plt.plot(fpr, tpr, 'g', label='Test')

plt.title('ROC Curve')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```



[4.0] Conclusion

In [63]:

```
x=PrettyTable()
x.field_names = ['Vectorizer', 'max_depth', 'min_samples_split', 'AUC']
x.add_row(['BOW', '50', '500', '0.831107'])
x.add_row(['TFIDF', '50', '500', '0.821965'])
x.add_row(['Avg W2V', '10', '500', '0.823780'])
x.add_row(['TFIDF-W2V', '10', '500', '0.784230'])
print(x)
```

```
+------------+-----------+-------------------+----------+
| Vectorizer | max_depth | min_samples_split |   AUC    |
```

BOW	50	500	0.831107
TFIDF	50	500	0.821965
Avg W2V	10	500	0.823780
TFIDF-W2V	10	500	0.784230