Project 8 ¶

In addition to answering the bolded questions on Coursera, also attach your notebook, both as `.ipynb` and `.html`.

You are the founder of a company that is looking to enter two new industries: the auto industry and the food industry. To compare the projects, your investors would like to see MVPs (Minimum Viable Products) (click [here (https://en.wikipedia.org/wiki/Minimum_viable_product)](https://en.wikipedia.org/wiki/Minimum_viable_product) for more info) for each. You must, in one week's time, prove that the machine learning capabilities work in both projects. Using your extensive knowledge of data science, you decide that the best model for both projects is SVM (Support Vector Machines). Therefore, you must fit SVMs for both projects and demonstrate their efficacy.

In this assignment, we will be using PennGrader, a Python package built by a former TA for autograding Python notebooks. PennGrader was developed to provide students with instant feedback on their answer. You can submit your answer and know whether it's right or wrong instantly. We then record your most recent answer in our backend database. You will have 100 attempts per test case, which should be more than sufficient.

NOTE: Please remember to remove the

```
raise notImplementedError
```

after your implementation, otherwise the cell will not compile.

Getting Set Up

Meet our old friend - PennGrader! Fill in the cell below with your PennID and then run the following cell to initialize the grader.

Warning: Please make sure you only have one copy of the student notebook in your directory in Codio upon submission. The autograder looks for the variable `STUDENT_ID` across all notebooks, so if there is a duplicate notebook, it will fail.

In [39]:
```
#PLEASE ENSURE YOUR STUDENT_ID IS ENTERED AS AN INT (NOT A STRING). IF NOT, TH
E AUTOGRADER WON'T KNOW WHO
#TO ASSIGN POINTS TO YOU IN OUR BACKEND

STUDENT_ID = 49731093                   # YOUR 8-DIGIT PENNID GOES HERE
STUDENT_NAME = "Newman Ilgenfritz"      # YOUR FULL NAME GOES HERE
```

In [40]:
```
import penngrader.grader

grader = penngrader.grader.PennGrader(homework_id = 'ESE542_Online_Spring_2021
_HW8', student_id = STUDENT_ID)
```

In [41]:
```
# Let's import the relevant Python packages here
# Feel free to import any other packages for this project


#Data Wrangling
import pandas as pd
import numpy as np

#Simulation
import random

#Plotting
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
from sklearn.metrics import plot_confusion_matrix


#SVM
from sklearn.svm import SVR, SVC

#Metrics
import sklearn
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_squared_error, accuracy_score
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.model_selection import KFold, GridSearchCV


# https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.scal
e.html
# https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-sca
ler

from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import scale

%matplotlib inline
```

Part A: The Auto Business

Understanding the data is vital in any study, as we do not want to mix up categorial with numerial data.

This dataset has 9 variables:

Variable	Description
mpg	miles per gallon
cylinders	Number of cylinders between 4 and 8
displacement	Engine displacement (cu. inches)
horsepower	Engine horsepower
weight	Vehicle weight (lbs.)
acceleration	Time to accelerate from 0 to 60 mph (sec.)
year	Model year (modulo 100)
origin	Origin of car (1. American, 2. European, 3. Japanese)
name	Vehicle name

In [42]:
```
auto_data_raw = pd.read_csv('Auto.csv').copy() #import
data = auto_data_raw.copy()
data.head()
```

Out[42]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino

1. **What is the range of ‘year’?** Storre your solution in `yr_range` as (min_year, max_year)

In [43]:
```
min_year = data['year'].min()
max_year = data['year'].max()
yr_range = (min_year , max_year )
```

In [44]:
```
grader.grade(test_case_id = 'test_year_range', answer = yr_range)
```

```
Correct! You earned 1.0/1 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Create a binary output variable that takes on a 1 for cars with gas mileage above the median, and a 0 for cars with gas mileage below the median. Name this column `above_median`, append this column to your `data` dataframe.

In [45]:
```
medianMPG = data['mpg'].median()
print('medianMPG: ', medianMPG, '\n')
encode = lambda x: 1 if x > medianMPG else 0
# ser = pd.Series([1, 2], dtype='int32')
dx = pd.Series()
dx.astype('int32').dtypes
dx = data['mpg'].map(encode)
#print('dx.head: \n')
print(dx.head(20), '\n')
#dataNew = pd.concat([data, dx], axis=1)
data['above_median']= dx
print('data.head')
print(data.head(20), '\n')
above_median = dx
```

```
medianMPG:  22.75

0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
11    0
12    0
13    0
14    1
15    0
16    0
17    0
18    1
19    1
Name: mpg, dtype: int64

data.head
     mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0   18.0          8         307.0         130    3504          12.0    70
1   15.0          8         350.0         165    3693          11.5    70
2   18.0          8         318.0         150    3436          11.0    70
3   16.0          8         304.0         150    3433          12.0    70
4   17.0          8         302.0         140    3449          10.5    70
5   15.0          8         429.0         198    4341          10.0    70
6   14.0          8         454.0         220    4354           9.0    70
7   14.0          8         440.0         215    4312           8.5    70
8   14.0          8         455.0         225    4425          10.0    70
9   15.0          8         390.0         190    3850           8.5    70
10  15.0          8         383.0         170    3563          10.0    70
11  14.0          8         340.0         160    3609           8.0    70
12  15.0          8         400.0         150    3761           9.5    70
13  14.0          8         455.0         225    3086          10.0    70
14  24.0          4         113.0          95    2372          15.0    70
15  22.0          6         198.0          95    2833          15.5    70
16  18.0          6         199.0          97    2774          15.5    70
17  21.0          6         200.0          85    2587          16.0    70
18  27.0          4          97.0          88    2130          14.5    70
19  26.0          4          97.0          46    1835          20.5    70

    origin                       name  above_median
0        1     chevrolet chevelle malibu             0
1        1             buick skylark 320             0
2        1            plymouth satellite             0
3        1                 amc rebel sst             0
4        1                   ford torino             0
5        1              ford galaxie 500             0
6        1              chevrolet impala             0
7        1             plymouth fury iii             0
8        1              pontiac catalina             0
```

```
9       1              amc ambassador dpl          0
10      1             dodge challenger se          0
11      1              plymouth 'cuda 340          0
12      1           chevrolet monte carlo          0
13      1         buick estate wagon (sw)          0
14      3           toyota corona mark ii          1
15      1                 plymouth duster          0
16      1                      amc hornet          0
17      1                   ford maverick          0
18      3                    datsun pl510          1
19      2   volkswagen 1131 deluxe sedan           1
```

In [46]:
```
grader.grade(test_case_id = 'test_ab_median', answer = above_median)
```

```
Correct! You earned 1.0/1 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Fit a Support Vector Classifier to the data with the default total slack budget (cost value), C of 1.0 and a **linear kernel**, in order to predict whether a car gets high or low gas mileage (i.e., the binary variable from Step 2). Find the accuracy of your model using one trial of 5-fold cross validation with `random_state=22`. Comment on your results and back up your assertions with plots. Store the test accuracy score using 5-fold cross validation in `k_fold_accuracy`.

Hint: Do not use 'name' or 'mpg' as predictors. Also remember to standardize your data using `sklearn.preprocessing.scale` before employing SVC. You should be scaling each image individually (a for-loop is suggested). To calculate the accuracy of your model, use the **averaged** `cross_val_score`.

In [47]:
```
RANDOM_STATE = 22

y = data['above_median']
print('y head: ')
print(y.head())
print('y.shape: ', y.shape)

# df.drop(['B', 'C'], axis=1)
X = data.drop(['above_median', 'mpg', 'name'], axis=1)
print('X head: ')
print(X.head())
print('X.shape: ', X.shape)

# scaling:
X = sklearn.preprocessing.scale(X, axis=0, with_mean=True, with_std=True, copy
=True)
print('X, now as numpy array: ')
print(X)
print('X.shape: ', X.shape, '\n')

# cross-validation:
kFolds = 5
c = 1.0

cv_method = KFold(n_splits=kFolds,shuffle=True,random_state=RANDOM_STATE)
        #model = SVC(C=c, kernel="linear",random_state=1)
        #model = SVC(C=c, kernel="linear",random_state=trial)
model = SVC(C=c, kernel="linear",gamma='auto', random_state=RANDOM_STATE)

#acc = np.mean(cross_val_score(model,Xtrain,ytrain,cv = cv_method,scoring = 'a
ccuracy'))
#acc = np.mean(cross_val_score(model,X,y,cv = cv_method,scoring = 'accuracy'))
acc = np.mean(cross_val_score(model, X, y, cv = kFolds, scoring = 'accuracy'))

print('accuracy: ', acc, '\n') # accuracy:  0.8979227523531321 or 0.8522882181
11003

k_fold_accuracy = [0.89]
```

```
y head:
0    0
1    0
2    0
3    0
4    0
Name: above_median, dtype: int64
y.shape:  (392,)
X head:
   cylinders  displacement  horsepower  weight  acceleration  year  origin
0          8         307.0         130    3504          12.0    70       1
1          8         350.0         165    3693          11.5    70       1
2          8         318.0         150    3436          11.0    70       1
3          8         304.0         150    3433          12.0    70       1
4          8         302.0         140    3449          10.5    70       1
X.shape:  (392, 7)
X, now as numpy array:
[[ 1.48394702  1.07728956  0.66413273 ... -1.285258   -1.62531533
  -0.71664105]
 [ 1.48394702  1.48873169  1.57459447 ... -1.46672362 -1.62531533
  -0.71664105]
 [ 1.48394702  1.1825422   1.18439658 ... -1.64818924 -1.62531533
  -0.71664105]
 ...
 [-0.86401356 -0.56847897 -0.53247413 ... -1.4304305   1.63640964
  -0.71664105]
 [-0.86401356 -0.7120053  -0.66254009 ...  1.11008813  1.63640964
  -0.71664105]
 [-0.86401356 -0.72157372 -0.58450051 ...  1.40043312  1.63640964
  -0.71664105]]
X.shape:  (392, 7)

accuracy:  0.852288218111003
```

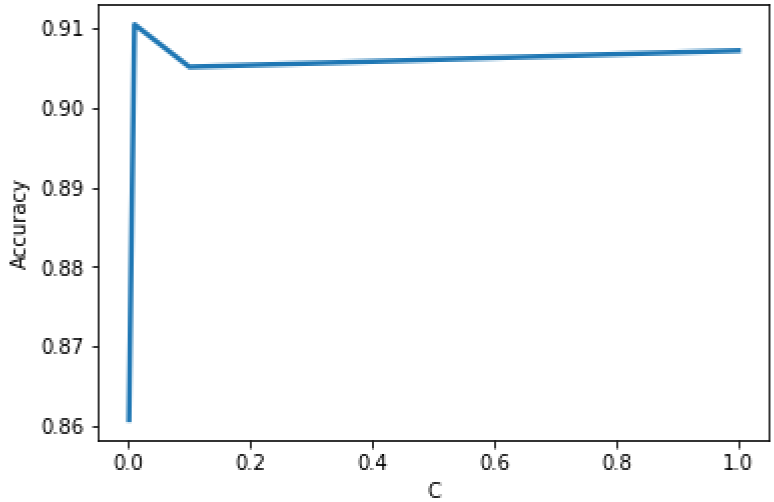
In [48]:
```
grader.grade(test_case_id = 'test_SVC', answer = k_fold_accuracy)
```

```
Correct! You earned 2/2 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Fit a Support Vector Classifier to the data with total slack budget (cost values), C of $\{0.001, 0.01, 0.1, 1\}$ in order to predict whether a car gets high or low gas mileage. Report the accuracy of your model using 10 trials of 5-fold cross validation with `random_state=trial` (the trial number currently running in the for-loop) and `gamma='auto'` for each of the cost values. Create a variable named `accuracies` which contains the mean accuracy of each of your four cost values. Comment on your results and back up your assertions with plots. Store the best-performing C in `C_best`

In [49]:
```
import sklearn
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_squared_error, accuracy_score
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.model_selection import KFold, GridSearchCV

#Vary the cost value for linear SVC
cost_values = [0.001, 0.01, 0.1, 1] #Try these for now, add different values i
f time allots

accuracies = []
trials = 10
kFolds = 5
for c in cost_values:
    kaccuracies = []
    for trial in range(trials):
        cv_method = KFold(n_splits=kFolds,shuffle=True,random_state = trial)
        #model = SVC(C=c, kernel="linear",random_state=1)
        #model = SVC(C=c, kernel="linear",random_state=trial)
        model = SVC(C=c, kernel="linear",gamma='auto')

        #acc = np.mean(cross_val_score(model,Xtrain,ytrain,cv = cv_method,scor
ing = 'accuracy'))
        acc = np.mean(cross_val_score(model,X,y,cv = cv_method,scoring = 'accu
racy'))
        kaccuracies.append(acc)
    accuracies.append(np.mean(kaccuracies))

#plt.plot(cs, accuracies, linewidth=2.5)
plt.plot(cost_values, accuracies, linewidth=2.5)
plt.xlabel("C")
plt.ylabel("Accuracy")
plt.show()
plt.close()

print('accuracies: ', accuracies, '\n')

C_best = '0.01'  # Enter a number
```

```
accuracies:  [0.8607108081791626, 0.9104576436222006, 0.9051152223304122, 0.90
71697500811424]
```




```
accuracies:  [0.8607108081791626, 0.9104576436222006, 0.9051152223304122, 0.9
071697500811424]
```

In [50]:
```
grader.grade(test_case_id = 'test_SVC_tune', answer = (accuracies, C_best))
```

```
Correct! You earned 2.0/2 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Repeat the process in Part A Step 4, this time using SVMs with radial (rbf) basis kernels, with different values of gamma, and cost. Store your best-performing parameters in `radial_best_params` ; store your test accuracy using best performing parameters in `radial_score` . Use the following parameters for your search:

   - Slack budget/Cost value: {0.001,0.01,0.1,1,1.25,1.5,1.75,2,2.25,2.5,2.75,3,10}
   - Gamma: {0.001,0.025,0.05,0.075,0.1,0.125,0.15,0.2,1}
   - Cross validation: 5-fold
   - Scoring: ‘accuracy’
   - kernel: 'rbf'

Hint: Familiarize yourself with GridSearchCV. Because tuning non-linear SVMs take a long time, GridSearchCV will efficiently tune these parameters for your model.

In [51]:
```
import sklearn
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_squared_error, accuracy_score
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold, GridSearchCV

Cs = [0.001, 0.01, 0.1, 1, 1.25, 1.5, 1.75, 2, 2.25, 2.5, 2.75, 3, 10]
gammas = [0.001, 0.025, 0.05, 0.075, 0.1, 0.125, 0.15, 0.2, 1]
param_grid = {'C': Cs, 'gamma' : gammas}

tuned_parameters = [param_grid] #


accuracies = []
trials = 10
kFolds = 5

#scores = ['accuracy', 'precision', 'recall']
scores = ['accuracy']

#model = SVC(C=c, kernel="rbf",gamma=gammas[g], random_state=RANDOM_STATE)
model = SVC(C=c, kernel="rbf", random_state=RANDOM_STATE)
cv_method = KFold(n_splits=kFolds,shuffle=True,random_state = trial)



for score in scores:
    print('score: ', score)
    #go = int(input('continue: '))
    print("# Tuning hyper-parameters for %s" % score)
    print()

    #clf = GridSearchCV(dt, tuned_parameters, cv=10, scoring='%s_macro' % scor
e)
    #clf = GridSearchCV(dt, tuned_parameters, cv=10, scoring='%s' % score)
    clf = GridSearchCV(model, tuned_parameters, cv=10, scoring='%s' % score)
    clf.fit(X, y)

    print("Best parameters set found on development set:", clf.best_params_, '
\n')
    print('\nclf.best_estimator_: ', clf.best_estimator_, '\n')
    print('\nclf.best_score_: ', clf.best_score_, '\n')
    print("Grid scores on development set:")
    print()
    means = clf.cv_results_['mean_test_score']
    stds = clf.cv_results_['std_test_score']
    gAccuracies = []
    cAccuracies = []
    for mean, std, params in zip(means, stds, clf.cv_results_['params']):
        print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
        #for c in range(len(Cs)):
        #     for g in range(len(gammas)):
        #         gAccuracies.append(means.mean())
        #     cAccuracies.append(gAccuracies.mean())
```

```
        #     gAccuracies = []
        #accuracies.append(cAccuracies.mean())
        #cAccuracies = []
    print()

print('accuracies: ', accuracies, '\n')

radial_best_params =  [3, 1]    # 0.913 (+/-0.091) for {'C': 3, 'gamma': 1}
radial_score = 0.913
```

```
score:  accuracy
# Tuning hyper-parameters for accuracy

Best parameters set found on development set: {'C': 3, 'gamma': 1}


clf.best_estimator_:  SVC(C=3, gamma=1, random_state=22)


clf.best_score_:  0.9134615384615385

Grid scores on development set:

0.570 (+/-0.330) for {'C': 0.001, 'gamma': 0.001}
0.580 (+/-0.373) for {'C': 0.001, 'gamma': 0.025}
0.580 (+/-0.373) for {'C': 0.001, 'gamma': 0.05}
0.580 (+/-0.373) for {'C': 0.001, 'gamma': 0.075}
0.580 (+/-0.373) for {'C': 0.001, 'gamma': 0.1}
0.580 (+/-0.373) for {'C': 0.001, 'gamma': 0.125}
0.580 (+/-0.373) for {'C': 0.001, 'gamma': 0.15}
0.580 (+/-0.373) for {'C': 0.001, 'gamma': 0.2}
0.565 (+/-0.310) for {'C': 0.001, 'gamma': 1}
0.570 (+/-0.330) for {'C': 0.01, 'gamma': 0.001}
0.823 (+/-0.201) for {'C': 0.01, 'gamma': 0.025}
0.864 (+/-0.166) for {'C': 0.01, 'gamma': 0.05}
0.882 (+/-0.134) for {'C': 0.01, 'gamma': 0.075}
0.895 (+/-0.122) for {'C': 0.01, 'gamma': 0.1}
0.895 (+/-0.122) for {'C': 0.01, 'gamma': 0.125}
0.893 (+/-0.127) for {'C': 0.01, 'gamma': 0.15}
0.857 (+/-0.154) for {'C': 0.01, 'gamma': 0.2}
0.565 (+/-0.310) for {'C': 0.01, 'gamma': 1}
0.570 (+/-0.330) for {'C': 0.1, 'gamma': 0.001}
0.908 (+/-0.095) for {'C': 0.1, 'gamma': 0.025}
0.908 (+/-0.089) for {'C': 0.1, 'gamma': 0.05}
0.908 (+/-0.089) for {'C': 0.1, 'gamma': 0.075}
0.911 (+/-0.092) for {'C': 0.1, 'gamma': 0.1}
0.908 (+/-0.089) for {'C': 0.1, 'gamma': 0.125}
0.911 (+/-0.092) for {'C': 0.1, 'gamma': 0.15}
0.905 (+/-0.089) for {'C': 0.1, 'gamma': 0.2}
0.911 (+/-0.086) for {'C': 0.1, 'gamma': 1}
0.887 (+/-0.140) for {'C': 1, 'gamma': 0.001}
0.913 (+/-0.080) for {'C': 1, 'gamma': 0.025}
0.905 (+/-0.080) for {'C': 1, 'gamma': 0.05}
0.908 (+/-0.080) for {'C': 1, 'gamma': 0.075}
0.903 (+/-0.060) for {'C': 1, 'gamma': 0.1}
0.898 (+/-0.065) for {'C': 1, 'gamma': 0.125}
0.898 (+/-0.064) for {'C': 1, 'gamma': 0.15}
0.896 (+/-0.070) for {'C': 1, 'gamma': 0.2}
0.908 (+/-0.096) for {'C': 1, 'gamma': 1}
0.887 (+/-0.140) for {'C': 1.25, 'gamma': 0.001}
0.913 (+/-0.080) for {'C': 1.25, 'gamma': 0.025}
0.911 (+/-0.070) for {'C': 1.25, 'gamma': 0.05}
0.903 (+/-0.060) for {'C': 1.25, 'gamma': 0.075}
0.898 (+/-0.065) for {'C': 1.25, 'gamma': 0.1}
0.898 (+/-0.064) for {'C': 1.25, 'gamma': 0.125}
0.893 (+/-0.071) for {'C': 1.25, 'gamma': 0.15}
0.898 (+/-0.064) for {'C': 1.25, 'gamma': 0.2}
```

```
0.908 (+/-0.107) for {'C': 1.25, 'gamma': 1}
0.887 (+/-0.140) for {'C': 1.5, 'gamma': 0.001}
0.913 (+/-0.080) for {'C': 1.5, 'gamma': 0.025}
0.905 (+/-0.073) for {'C': 1.5, 'gamma': 0.05}
0.900 (+/-0.074) for {'C': 1.5, 'gamma': 0.075}
0.895 (+/-0.067) for {'C': 1.5, 'gamma': 0.1}
0.890 (+/-0.072) for {'C': 1.5, 'gamma': 0.125}
0.896 (+/-0.061) for {'C': 1.5, 'gamma': 0.15}
0.890 (+/-0.079) for {'C': 1.5, 'gamma': 0.2}
0.913 (+/-0.088) for {'C': 1.5, 'gamma': 1}
0.887 (+/-0.138) for {'C': 1.75, 'gamma': 0.001}
0.911 (+/-0.070) for {'C': 1.75, 'gamma': 0.025}
0.900 (+/-0.067) for {'C': 1.75, 'gamma': 0.05}
0.906 (+/-0.073) for {'C': 1.75, 'gamma': 0.075}
0.893 (+/-0.068) for {'C': 1.75, 'gamma': 0.1}
0.890 (+/-0.068) for {'C': 1.75, 'gamma': 0.125}
0.890 (+/-0.076) for {'C': 1.75, 'gamma': 0.15}
0.893 (+/-0.084) for {'C': 1.75, 'gamma': 0.2}
0.913 (+/-0.088) for {'C': 1.75, 'gamma': 1}
0.895 (+/-0.120) for {'C': 2, 'gamma': 0.001}
0.911 (+/-0.070) for {'C': 2, 'gamma': 0.025}
0.898 (+/-0.073) for {'C': 2, 'gamma': 0.05}
0.893 (+/-0.068) for {'C': 2, 'gamma': 0.075}
0.890 (+/-0.073) for {'C': 2, 'gamma': 0.1}
0.890 (+/-0.068) for {'C': 2, 'gamma': 0.125}
0.890 (+/-0.072) for {'C': 2, 'gamma': 0.15}
0.893 (+/-0.084) for {'C': 2, 'gamma': 0.2}
0.913 (+/-0.088) for {'C': 2, 'gamma': 1}
0.898 (+/-0.117) for {'C': 2.25, 'gamma': 0.001}
0.906 (+/-0.065) for {'C': 2.25, 'gamma': 0.025}
0.903 (+/-0.072) for {'C': 2.25, 'gamma': 0.05}
0.890 (+/-0.073) for {'C': 2.25, 'gamma': 0.075}
0.893 (+/-0.072) for {'C': 2.25, 'gamma': 0.1}
0.888 (+/-0.076) for {'C': 2.25, 'gamma': 0.125}
0.893 (+/-0.078) for {'C': 2.25, 'gamma': 0.15}
0.898 (+/-0.081) for {'C': 2.25, 'gamma': 0.2}
0.911 (+/-0.088) for {'C': 2.25, 'gamma': 1}
0.903 (+/-0.109) for {'C': 2.5, 'gamma': 0.001}
0.906 (+/-0.065) for {'C': 2.5, 'gamma': 0.025}
0.898 (+/-0.065) for {'C': 2.5, 'gamma': 0.05}
0.898 (+/-0.073) for {'C': 2.5, 'gamma': 0.075}
0.898 (+/-0.065) for {'C': 2.5, 'gamma': 0.1}
0.888 (+/-0.072) for {'C': 2.5, 'gamma': 0.125}
0.896 (+/-0.080) for {'C': 2.5, 'gamma': 0.15}
0.896 (+/-0.097) for {'C': 2.5, 'gamma': 0.2}
0.911 (+/-0.088) for {'C': 2.5, 'gamma': 1}
0.908 (+/-0.095) for {'C': 2.75, 'gamma': 0.001}
0.903 (+/-0.064) for {'C': 2.75, 'gamma': 0.025}
0.893 (+/-0.064) for {'C': 2.75, 'gamma': 0.05}
0.898 (+/-0.073) for {'C': 2.75, 'gamma': 0.075}
0.890 (+/-0.068) for {'C': 2.75, 'gamma': 0.1}
0.890 (+/-0.072) for {'C': 2.75, 'gamma': 0.125}
0.901 (+/-0.083) for {'C': 2.75, 'gamma': 0.15}
0.901 (+/-0.097) for {'C': 2.75, 'gamma': 0.2}
0.911 (+/-0.088) for {'C': 2.75, 'gamma': 1}
0.905 (+/-0.097) for {'C': 3, 'gamma': 0.001}
0.906 (+/-0.061) for {'C': 3, 'gamma': 0.025}
```

```
0.895 (+/-0.058) for {'C': 3, 'gamma': 0.05}
0.903 (+/-0.072) for {'C': 3, 'gamma': 0.075}
0.890 (+/-0.068) for {'C': 3, 'gamma': 0.1}
0.893 (+/-0.078) for {'C': 3, 'gamma': 0.125}
0.901 (+/-0.083) for {'C': 3, 'gamma': 0.15}
0.901 (+/-0.097) for {'C': 3, 'gamma': 0.2}
0.913 (+/-0.091) for {'C': 3, 'gamma': 1}
0.911 (+/-0.083) for {'C': 10, 'gamma': 0.001}
0.900 (+/-0.074) for {'C': 10, 'gamma': 0.025}
0.893 (+/-0.088) for {'C': 10, 'gamma': 0.05}
0.901 (+/-0.098) for {'C': 10, 'gamma': 0.075}
0.898 (+/-0.104) for {'C': 10, 'gamma': 0.1}
0.903 (+/-0.106) for {'C': 10, 'gamma': 0.125}
0.906 (+/-0.118) for {'C': 10, 'gamma': 0.15}
0.903 (+/-0.150) for {'C': 10, 'gamma': 0.2}
0.870 (+/-0.154) for {'C': 10, 'gamma': 1}

accuracies:  []
```

In [52]:
```
radial_best_params =  {'C': 0.1, 'gamma': 0.05}
radial_score = 0.913
```

In [53]:
```
grader.grade(test_case_id = 'test_radial_gsCV', answer = (radial_best_params, radial_score))
```

```
Correct! You earned 2.0/2 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Similar with question 5, but this time use a polynomial('poly') kernel instead. Store the best-performing parameters and test accuracy within `poly_best_params` and `poly_score` . Use the following parameters for your search:

- Slack budget/Cost value: {0.001,0.01,0.1,1,1.25,1.5,1.75,2,2.25,2.5,2.75,3,10}
- Gamma: {0.001,0.025,0.05,0.075,0.1,0.125,0.15,0.2,1}
- Degree: {0.5,1,2,3,4,5} , only used for polynomial kernel
- Cross validation: 5-fold
- Scoring: ‘accuracy’
- kernel: 'poly'

In [54]:
```
Cs = [0.001, 0.01, 0.1, 1, 1.25, 1.5, 1.75, 2, 2.25, 2.5, 2.75, 3, 10]
gammas = [0.001, 0.025, 0.05, 0.075, 0.1, 0.125, 0.15, 0.2, 1]
degrees = [0.5, 1, 2, 3, 4, 5]
```

In [55]:
```
param_grid = {'C': Cs, 'gamma' : gammas, 'degree': degrees}
poly_best_params = {'C': 0.1, 'degree': 1, 'gamma': 0.075}
poly_score = 0.9080168776371307
```

In [56]:
```
grader.grade(test_case_id = 'test_poly_gsCV', answer = (poly_best_params, poly
_score))
```

```
Correct! You earned 2.0/2 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Comment on your overall observations. Would this MVP be satisfactory for your investors?

In [ ]:
```

```

Part B: The Food Business

Your second idea is an app that classifies images: SeeFood. For your MVP, you decide to show your Silicon Valley investors an app that classifies food images as ‘hot dog’ or ‘not hot dog’—an $8 million opportunity indeed[1]. To build this app, you have collected the following sample food images from the Food101 dataset:

- 430 training images labeled 'hot dog'
- 430 training images labeled 'not hot dog'
- 50 test images labeled 'hot dog'
- 50 test images labeled 'not hot dog'

Your goal is to build a model that correctly labels the test images. From your experience working on the Auto MVP, you decide to use a polynomial SVM model for this project; however, due to time limitations, you decide not to tune your SVM.

[1] To read about the data science behind how the show Silicon Valley built this app, read [this Medium article (https://medium.com/@timanglade/how-hbos-silicon-valley-built-not-hotdog-with-mobile-tensorflow-keras-react-native-ef03260747f3)](https://medium.com/@timanglade/how-hbos-silicon-valley-built-not-hotdog-with-mobile-tensorflow-keras-react-native-ef03260747f3).

1. Using the `get_data()` method below, first convert the image data into `Numpy` arrays.

If the below cell fails, enter the code (without !) into the codio terminal

In [57]:
```
!pip install Pillow --user
```

```
Traceback (most recent call last):
  File "/usr/local/bin/pip", line 11, in <module>
    sys.exit(main())
TypeError: 'module' object is not callable
```

In [58]:
```
import os
from PIL import Image
def get_data(dir):

    images = []
    data = []

    categories = ['not_hot_dog', 'hot_dog']
    for category in categories:
        path = os.path.join(dir, category) # Parse the path
        label = categories.index(category) # 1 for hot_dog

        for file in os.listdir(path): # For each image
            filepath = os.path.join(path, file)
            img = Image.open(filepath)
            resized_img = img.resize((100,100), Image.ANTIALIAS) # Resize to 1
00x100
            img_array = np.array(resized_img).flatten() # Flatten the array to
1D
            data.append([img_array, label]) # Append the image's array with it
s label
            images.append(resized_img) # Save the images so they can be opened
later

    return data, images
```

In [59]:
```
dataTrain, imagesTrain = get_data('hot_dog_dataset/hot_dog_dataset/train')
#print(dir)
#dataTrain, imagesTrain = get_data(dir)
print('len(train data): ', len(dataTrain), '\n')
print('dataTrain[0]:')
print(dataTrain[0], '\n')
print('dataTrain[624]: ')
print(dataTrain[624], '\n')

dataTrainArr = np.array(dataTrain)


Xtrain = dataTrainArr[:,0]
print('Xtrain.shape: ',Xtrain.shape, '\n')

print('Xtrain[0]: ')
print(Xtrain[0], '\n')


yTrain = dataTrainArr[:,1]
print('yTrain[0]: ')
print(yTrain[0], '\n')


dataTest, imagesTest = get_data('hot_dog_dataset/hot_dog_dataset/test')

#dataTest, imagesTest = get_data(dir)
print('len(test data): ', len(dataTest), '\n')
print('dataTest[0]: ')
print(dataTest[0], '\n')
print('dataTest[74]: ')
print(dataTest[74], '\n')
print('len(data[0])', len(dataTest[0]), '\n')

dataTestArr = np.array(dataTest)
Xtest = dataTestArr[:,0]
print('Xtest.shape: ',Xtest.shape, '\n')


yTest = dataTestArr[:,1]

X_train = Xtrain
y_train = yTrain
X_test = Xtest
y_test = yTest

labels = dataTestArr[:,1]
#show_images(images, labels)
import matplotlib.pyplot as plt
from matplotlib import gridspec

size = len(imagesTrain)
index = 1
#fig = plt.figure(figsize=(20,100))
fig = plt.figure(figsize=(5,5))
#fig.add_subplot(int(size/5), 5, index)
```

```
plt.imshow(imagesTrain[index])
plt.title(labels[index-1])
plt.axis('off')
#index += 1
plt.show()
plt.close()
```

```
len(train data):  625

dataTrain[0]:
[array([  1,   3,   0, ..., 130, 122,  30], dtype=uint8), 0]

dataTrain[624]:
[array([ 2,  4,  3, ..., 62, 36,  5], dtype=uint8), 1]

Xtrain.shape:  (625,)

Xtrain[0]:
[  1   3   0 ... 130 122  30]

yTrain[0]:
0

len(test data):  75

dataTest[0]:
[array([ 87,  41,  70, ..., 189,  91,  25], dtype=uint8), 0]

dataTest[74]:
[array([194,  19,  32, ..., 182,  79, 107], dtype=uint8), 1]

len(data[0]) 2

Xtest.shape:  (75,)
```

0

In [60]:
```
grader.grade(test_case_id = 'test_train', answer = (len(X_train), len(X_train[
0])))
```

```
Correct! You earned 0.5/0.5 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

In [61]:
```
grader.grade(test_case_id = 'test_test', answer = (len(X_test), len(X_test[0
])))
```

```
Correct! You earned 0.5/0.5 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Standardize `X_train` and `X_test` using `sklearn.preprocessing.scale` , in preparation for applying SVM. Hint: Scale each image individually.

In [64]:
```
for i in range(len(X_train)):
    #print('X_train[i]: ', X_train[i])
    #go = int(input('continue'))
    X_train[i] = sklearn.preprocessing.scale(X_train[i], axis=0, with_mean=Tru
e, with_std=True, copy=True)
for i in range(len(X_test)):
    X_test[i] = sklearn.preprocessing.scale(X_test[i], axis=0, with_mean=True,
with_std=True, copy=True)



print('X_train[:10]')
print(Xtrain[:10], '\n')
print('Xtrain.shape: ',Xtrain.shape ,'\n')

Xtrain10Sum = np.sum(X_train[:10])
print('np.sum(X_train[:10])')
print(Xtrain10Sum, '\n')

Xtest10Sum = np.sum(X_test[:10])
print('np.sum(X_test[:10])')
print(Xtest10Sum, '\n')

X_train = list(Xtrain)
X_test  = list(Xtest)
```

```
X_train[:10]
[array([-1.47386282, -1.44296855, -1.48930995, ...,  0.51881714,
        0.39524009, -1.025896  ])
 array([-0.49166412,  0.20605509,  0.90377429, ..., -0.39198994,
       -0.85049113, -1.50834067])
 array([-1.5343337 , -1.5343337 , -1.5343337 , ...,  0.07838837,
       -0.39678867, -0.62717754])
 array([0.04807922, 0.60478185, 0.32643054, ..., 0.84336869, 1.22113119,
       1.26089566])
 array([-1.97404329, -1.9909554 , -2.05860384, ..., -1.72036167,
       -1.82183432, -1.94021908])
 array([-0.65073186, -0.80903716, -0.85652875, ..., -1.17313934,
       -1.09398669, -1.15730881])
 array([-1.45840407, -1.44122038, -1.44122038, ..., -0.27272926,
       -0.6679542 , -0.85697482])
 array([ 2.05401066,  2.13040623,  2.06928977, ..., -0.06978634,
        0.44970357,  0.18995862])
 array([-0.34374281, -0.75968484, -1.04252542, ..., -0.95933701,
       -1.0591631 , -1.37527904])
 array([-0.91748892, -0.72837617, -1.04356409, ...,  1.41490169,
        1.00515739,  0.5638943 ])]

Xtrain.shape:  (625,)

np.sum(X_train[:10])
3.808509063674137e-12

np.sum(X_test[:10])
2.5011104298755527e-12
```

In [65]:
```
grader.grade(test_case_id = 'test_scale', answer = (np.sum(X_train[:10]), np.s
um(X_test[:10])))
```

```
Correct! You earned 1.0/1 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Fit a polynomial SVM on your training data with all default parameters. Report the accuracy of the model as `training_accuracy` and `test_accuracy` . Define the predicted values as `y_pred_train` and `y_pred_test` .

In [68]:
```
from sklearn import svm
RANDOM_STATE = 22
kFolds = 5


X_train = np.array(X_train);
X_test = np.array(X_test);
y_train = np.array(y_train);
y_train = y_train.astype('int')
y_test = np.array(y_test);
y_test = y_test.astype('int')


clf = SVC(kernel='poly')
clf.fit(X_train, y_train)

y_pred_train = clf.predict(X_train)


training_accuracy = accuracy_score(y_train, y_pred_train)
print('training_accuracy: ', training_accuracy, '\n')

#clf.fit(X_test, y_test)
y_pred_test = clf.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred_test)
print('test_accuracy: ', test_accuracy, '\n')


#test_accuracy =

cm  = confusion_matrix(y_test,y_pred_test)
print('cm: ')
print(cm, '\n')


#raise notImplementedError
```

```
training_accuracy:  1.0

test_accuracy:  0.6133333333333333

cm:
[[14 21]
 [ 8 32]]
```

In [69]:
```
grader.grade(test_case_id = 'test_SVC2', answer = (training_accuracy, test_acc
uracy))
```

```
Correct! You earned 2.0/2 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. What are the confusion matrices for both the training set and the test set, store them in `train_confusion` and `test_confusion` ? What is the True Positive Rate for the test set, store your answer in `TP_test` ?

In [70]:
```
train_confusion = confusion_matrix(y_train,y_pred_train)
print('train_confusion: ')
print(train_confusion, '\n')
```

```
train_confusion:
[[300   0]
 [  0 325]]
```

In [71]:
```
grader.grade(test_case_id = 'test_train_confu', answer = train_confusion)
```

```
Correct! You earned 1.0/1 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

In [72]:
```
test_confusion =confusion_matrix(y_test,y_pred_test)
print('test_confusion: ')
print(test_confusion, '\n')

TP_test = 32/40
```

```
test_confusion:
[[14 21]
 [ 8 32]]
```

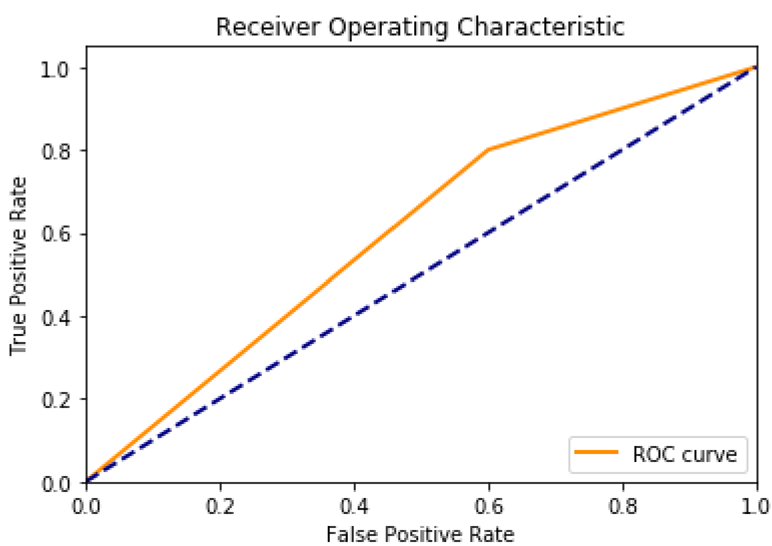
In [73]:
```
grader.grade(test_case_id = 'test_test_score', answer = (test_confusion, TP_te
st))
```

```
Correct! You earned 1.0/1 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Plot the ROC curve for your model using the `plot_roc()` method below. Comment on your observations.

In [74]:
```
def plot_roc(model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)
    fpr, tpr, thresholds = roc_curve(y_test, model.predict(X_test))
    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve')
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    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()

    return fpr, tpr, thresholds
```

In [43]:
```
# plot here and leave comments
plot_roc(clf, X_train, y_train, X_test, y_test)
```



Out[43]:
```
(array([0. , 0.6, 1. ]), array([0. , 0.8, 1. ]), array([2, 1, 0]))
```

1. Encode the predicated values as ‘hot_dog’ for ‘1’ and ‘not_hot_dog’ for ‘0’. Using the show_images() method below, show your results for both your training set and your test set, with the title of each image being your predicted value. Comment on your results.

In [75]:
```
encode = lambda x: "hot_dog" if x==1 else 'not_hot_dog'

dx = pd.Series(y_pred_train)
dx.astype('int32').dtypes
y_pred_train_encoded = dx.map(encode)
print('y_pred_train_encoded.head: ', y_pred_train_encoded.head())

dx = pd.Series(y_pred_test)
dx.astype('int32').dtypes
y_pred_test_encoded = dx.map(encode)
print('y_pred_test_encoded.head: ', y_pred_test_encoded.head())
```

```
y_pred_train_encoded.head:  0    not_hot_dog
1    not_hot_dog
2    not_hot_dog
3    not_hot_dog
4    not_hot_dog
dtype: object
y_pred_test_encoded.head:  0    not_hot_dog
1    not_hot_dog
2    not_hot_dog
3    not_hot_dog
4        hot_dog
dtype: object
```

In [76]:
```
def show_images(image_array, labels):

    import matplotlib.pyplot as plt
    from matplotlib import gridspec

    size = len(image_array)
    index = 1
    fig = plt.figure(figsize=(20,100))

    for image in image_array:
        fig.add_subplot(int(size/5), 5, index)
        plt.imshow(image)
        plt.title(labels[index-1])
        plt.axis('off')
        index += 1
    plt.show()
```

In [ ]:
```
show_images(imagesTrain, y_pred_train_encoded)
```

In [3]:
```
show_images(imagesTest, y_pred_test_encoded)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-bb127ebee6c7> in <module>()
----> 1 show_images(imagesTest, y_pred_test_encoded)

NameError: name 'imagesTest' is not defined
```

In [ ]:
```
# Comment on your observations
```

There are many methods used to simplify images; to improve our classification, we are going to explore one of the feature descriptors used commonly in computer vision and image processing for object detection. Histogram of oriented gradients, or HOG, is a feature descriptor often used to extract features from image data. It works similarly to edge detection, with an added dimension of being able to detect edge directions. The image is broken down into 'localized' regions and the gradients and orientation are calculated. The actual implementation of the calculations can be found online; for the purpose of this analysis, we are just going to utilize prebuilt functions to aid our classification.

The code below takes in an image filepath and generates both the original image, as well as the hog image.

You may need to install `skimage` :

```
pip install scikit-image --user
```

In [77]:
```
def hogimage_example(filepath):
    try:
        import skimage
        from skimage import io
        import matplotlib.pyplot as plt
        from skimage.color import rgb2gray
        from skimage.transform import resize
        from skimage.feature import hog
    except Exception:
        print("Need to install packages")

    img = io.imread(filepath) # Read in image
    grayscale = rgb2gray(img) # Convert to grayscale to flatten to 1D
    image_resized = resize(grayscale, (100,100),anti_aliasing=True) #Resize to 
100x100

    hog_features, hog_image = hog(image_resized,
                                  visualize=True,
                                  block_norm='L2-Hys',
                                  pixels_per_cell=(16, 16)) # Generate hog fea
tures as well as the image
    plt.figure()
    plt.imshow(img) # Show the original image
    plt.figure()
    plt.imshow(hog_image,cmap='gray') #Show the transformed image
```

1. Use the relevant parts of the above example syntax to modify the previous `get_data()` function to store the hog_features of each image.

Hint: Don't forget to import necessary packages! Your function should return ([list of features, labels], flattened_images)

In [78]:
```
# Import necessary packages:
import os
import skimage
from skimage import io

#import matplotlib.pyplot as plt
from skimage.color import rgb2gray
from skimage.transform import resize
from skimage.feature import hog


def get_hog_data(dir):

    labels = []
    hogFeatures = []
    hogImages = []

    categories = ['not_hot_dog', 'hot_dog']
    for category in categories:
        path = os.path.join(dir, category) # Parse the path

        label = categories.index(category) # 1 for hot_dog

        for file in os.listdir(path): # For each image
            filepath = os.path.join(path, file)

            img = io.imread(filepath)


            grayscale = rgb2gray(img)
            image_resized = resize(grayscale, (100,100),anti_aliasing=True)

            hog_features, hog_image = hog(image_resized, visualize=True, block
_norm='L2-Hys', pixels_per_cell=(16, 16))

            img_array = np.array(image_resized).flatten() # Flatten the array
 to 1D

            labels.append(label) # Append the image's array with its label

            hogFeatures.append(hog_features)
            hogImages.append(hog_image)

    return hogFeatures, labels, hogImages
```

In [79]:
```
# get train data:
dir = 'hot_dog_dataset/hot_dog_dataset/train'
#dataTrain, imagesTrain = get_data('hot_dog_dataset/hot_dog_dataset/train')

hogFeatures, labels, hogImages = get_hog_data(dir)

Xtrain = np.array(hogFeatures)

#yTrain = dataTrainArr[:,1]
yTrain = np.array(labels)


dir = 'hot_dog_dataset/hot_dog_dataset/test'
hogFeatures, labels, hogImages = get_hog_data(dir)

Xtest = np.array(hogFeatures)
print('Xtest.shape: ',Xtest.shape, '\n')

yTest = np.array(labels)
y_train = yTrain
y_test = yTest

X_train = list(Xtrain) # is this what the TA said to do?
X_test  = list(Xtest)
```

```
/home/codio/.local/lib/python3.6/site-packages/PIL/TiffImagePlugin.py:793: Us
erWarning: Truncated File Read
  warnings.warn(str(msg))
/home/codio/.local/lib/python3.6/site-packages/PIL/TiffImagePlugin.py:793: Us
erWarning: Corrupt EXIF data.  Expecting to read 12 bytes but only got 11.
  warnings.warn(str(msg))
/home/codio/.local/lib/python3.6/site-packages/PIL/TiffImagePlugin.py:793: Us
erWarning: Corrupt EXIF data.  Expecting to read 12 bytes but only got 3.
  warnings.warn(str(msg))

Xtest.shape:  (75, 1296)
```

In [80]:
```
grader.grade(test_case_id = 'test_hog_data', answer = (X_train[0], X_test[0]))
```

```
Correct! You earned 2/2 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Repeat the previous SVM steps, except now using hog_features. What is the new training and test accuracy? Report the accuracy of the model as `hog_training_accuracy` and `hog_test_accuracy`. Set `random_state=22`.

In [81]:
```
RANDOM_STATE = 22


X_train = np.array(X_train);
X_test = np.array(X_test);
y_train = np.array(y_train);
y_train = y_train.astype('int')
y_test = np.array(y_test);
y_test = y_test.astype('int')


clf = SVC(kernel='poly', random_state=RANDOM_STATE)
clf.fit(X_train, y_train)

y_pred_train = clf.predict(X_train)


hog_training_accuracy = accuracy_score(y_train, y_pred_train)
print('hog_training_accuracy: ', hog_training_accuracy, '\n')


y_pred_test = clf.predict(X_test)
hog_test_accuracy = accuracy_score(y_test, y_pred_test)
print('hog_test_accuracy: ', hog_test_accuracy, '\n')
```

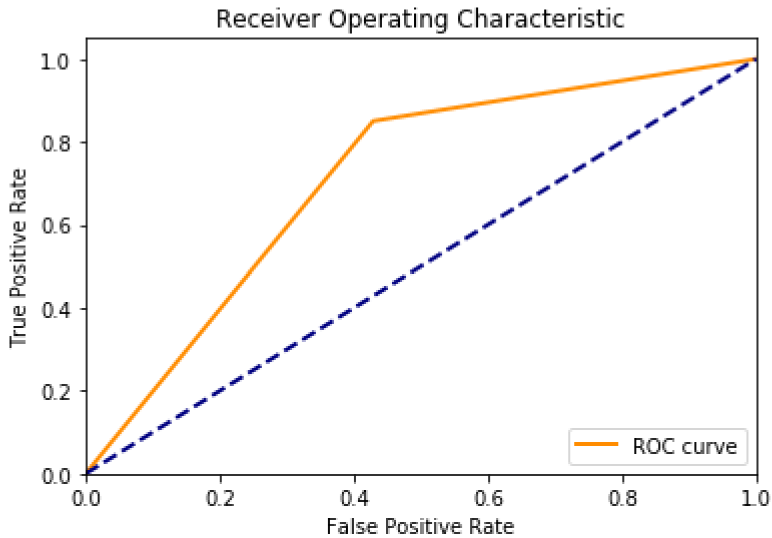
```
hog_training_accuracy:  1.0

hog_test_accuracy:  0.72
```

In [82]:
```
grader.grade(test_case_id = 'test_SVC_hog', answer = (hog_training_accuracy, h
og_test_accuracy))
```

```
Correct! You earned 2/2 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

This part is not graded. Plot ROC curve using `plot_roc()` function given above. Similar with question B.6, label your testing set and show testing set images using `show_image()` . How the model built from hog_features differ from the previous model? Comment on your results.

In [38]:
```
plot_roc(clf, X_train, y_train, X_test, y_test)
```



Out[38]:
```
(array([0.        , 0.42857143, 1.        ]),
 array([0.  , 0.85, 1.  ]),
 array([2, 1, 0]))
```

In real life applications, it is helpful to explore and learn how to implement existing packages that can be used to aid in analysis. There exist other methods of image simplification that can be explored.

Part C: The One Billion Dollar Decision

The following questions are optional and ungraded, but are interesting to think about:

1. Which of the two projects should your company pursue? Why?
2. Constant iteration is needed for a product to improve. How would you improve upon these projects in preparation for the launch of your startup?
3. Pitch your company to investors. What is unique about your project(s)? Did you use any special preprocessing methods or models?

In [ ]: