A2 - Python

This assignment will cover topics of classification.

Make sure that you keep this notebook named as "a2.ipynb"

Any other packages or tools, outside those listed in the assignments or Canvas, should be cleared by Dr. Brown before use in your submission.

Q0 - Setup

The following code looks to see whether your notebook is run on Gradescope (GS), Colab (COLAB), or the linux Python environment you were asked to setup.

In [1]:
```
import re
import os
import platform
import sys

# flag if notebook is running on Gradescope
if re.search(r'amzn', platform.uname().release):
    GS = True
else:
    GS = False

# flag if notebook is running on Colaboratory
try:
  import google.colab
  COLAB = True
except:
  COLAB = False

# flag if running on Linux lab machines.
cname = platform.uname().node
if re.search(r'(guardian|colossus|c28)', cname):
    LLM = True
else:
    LLM = False

print("System: GS - %s, COLAB - %s, LLM - %s" % (GS, COLAB, LLM))
```

```
System: GS - False, COLAB - False, LLM - True
```

Notebook Setup

It is good practice to list all imports needed at the top of the notebook. You can import modules in later cells as needed, but listing them at the top clearly shows all which are needed to be available / installed.

If you are doing development on Colab, the otter-grader package is not available, so you will need to install it with pip (uncomment the cell directly below).

In [2]:
```
# Only uncomment if you developing on Colab
# if COLAB == True:
#     print("Installing otter:")
#     !pip install otter-grader==4.2.0
```

In [3]:
```
# Import standard DS packages
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import math
import statistics
import textwrap
%matplotlib inline

from sklearn import tree        # decision tree classifier
from sklearn import neighbors   # knn classifier
from sklearn import naive_bayes # naive bayes classifier
from sklearn import svm         # svm classifier
from sklearn import ensemble    # ensemble classifiers
from sklearn import metrics     # performance evaluation metrics
from sklearn import model_selection
from sklearn import preprocessing
from sklearn import pipeline
# import graphviz, pydotplus

from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.metrics import f1_score, roc_auc_score,mean_squared_error
from sklearn.metrics import confusion_matrix

# Package for Autograder
import otter
grader = otter.Notebook()

from warnings import simplefilter
simplefilter(action='ignore', category=FutureWarning)
```

In [6]:
```
grader.check("q0")
```

Out[6]: **q0** passed! 💯

Q1 - Exploratory Data Analysis

Consider the `movies` data set available on Canvas. The data set is made up of over 600 randomly selected movies, released before 2016, with information extracted from IMDB and Rotten Tomatoes. A code book on the variables is also provided.

You should explore the files a bit in a text editor to understand the format. The variables are made up of different types: nominal, ordinal, numeric, etc. We will refer to the different variables by their column / codebook names.

Q1(a) - Examine data for loading

Look at the `movies` data set. Is there any missing data in the `movies` data set? If so, how is it encoded?

Yes, Missing data is encoded as: 'NA'

Q1(b) - Load the data

Load the movies data into a DataFrame `q1movies`. Is there any missing data in the `movies` data set? If yes, make sure to encode those missing values when reading the data in pandas `read_csv` function.

In [7]:
```
# Read in movies data with pandas "read_csv" function
#  Use column names from the original csv file

na_values = ["N/A"]
q1movies = pd.read_csv("data/movies.csv", na_values=na_values)
q1movies.head()
```

Out[7]:

	title	title_type	genre	runtime	mpaa_rating	studio	thtr_rel_year	thtr_rel_month	thtr_rel_day	dvd_rel_year	...	best_(
0	Filly Brown	Feature Film	Drama	80.0	R	Indomina Media Inc.	2013	4	19	2013.0	...	
1	The Dish	Feature Film	Drama	101.0	PG-13	Warner Bros. Pictures	2001	3	14	2001.0	...	
2	Waiting for Guffman	Feature Film	Comedy	84.0	R	Sony Pictures Classics	1996	8	21	2001.0	...	
3	The Age of Innocence	Feature Film	Drama	139.0	PG	Columbia Pictures	1993	10	1	2001.0	...	
4	Malevolence	Feature Film	Horror	90.0	R	Anchor Bay Entertainment	2004	9	10	2005.0	...	

5 rows × 32 columns

In [8]:
```
grader.check("q1b")
```

Out[8]: **q1b** passed! 🌈

Q1(c) - Clean data

We want to clean up data with respect to the missing values.

Ignore any missing values in the `studio` , `dvd_rel_year` , `dvd_rel_month` , `dvd_rel_day` , and all variables including and listed after `best_pic_nom` . For other missing values, remove the sample that contains the missing value.

Save the resulting DataFrame in the `movies` variable.

In [9]:
```

# Ignore missing values in "studio", "dvd_rel_year", "dvd_rel_month",
#   "dvd_rel_day", and all varaibles including and after "best_pic_nom"
# For other missing values, remove the sample that contains the missing value.

movies = q1movies[q1movies['runtime'].isnull() == False]

movies.shape
```

Out[9]: (645, 32)

In [10]:
```
grader.check("q1c")
```

Out[10]: **q1c** passed! 🚀

Q1(d) - Statistics, part 1

For the following variables, report out a five number summary: `critics_score` and `runtime`

Store results in a DataFrame: `q1d`

Hint: consider using the `describe` function.

In [11]:
```
# Report five number summary for variables `critics_score` and `runtime` in
#  a DataFrame "q1d"
#  Rows should represent: min, Q1 - 25%, Q2 - 50%, Q3 - 50%, max
#  Columns should be `critics_score` then `runtime`

runtime = movies['runtime'].describe()[3:].values
critics_score = movies['critics_score'].describe()[3:].values
idx = ['min', 'Q1 - 25%', 'Q2 - 50%', 'Q3 - 75%', 'max']

data =  {'critics_score': critics_score,'runtime': runtime}

q1d = pd.DataFrame(data,index=idx)
q1d
```

Out[11]:

	critics_score	runtime
min	1.0	39.0
Q1 - 25%	33.0	92.0
Q2 - 50%	61.0	103.0
Q3 - 75%	83.0	116.0
max	100.0	267.0

In [12]:
```
grader.check("q1d")
```

Out[12]: **q1d** passed! 🌟

Q1(e) - Statistics, part 2

Report the mean, median, and mode of `audience_score` and `imdb_rating` to the given variables.

In [13]:
```
# Report mean, median and mode of "audience_score" and "imdb_rating"

# For audience_score-
q1e_as_mean = movies['audience_score'].mean()
q1e_as_median = movies['audience_score'].median()
q1e_as_mode = movies['audience_score'].mode()[0]

# For imdb_rating-
q1e_ir_mean = movies['imdb_rating'].mean()
q1e_ir_median = movies['imdb_rating'].median()
q1e_ir_mode = movies['imdb_rating'].mode()[0]
```

In [14]:
```
grader.check("q1e")
```

Out[14]: **q1e** passed! 🎉

Q1(f) - Statistics, part 3

Report the first quartile, 37th percentile, third quartile, and 83rd percentile for `audience_score` and `imdb_rating` and assign it to the given variables.

In [15]:
```
movies['audience_score'].quantile([0.25]).iloc[0]
```

Out[15]:
```
46.0
```

In [16]:
```
# Report first quartile, 31st percentile, third quartile, and 90th percentile
#  of "audience_score" and "imdb_rating"

# For audience_score-
q1f_as_q1 = movies['audience_score'].quantile([0.25]).iloc[0]
q1f_as_p37=movies['audience_score'].quantile([0.37]).iloc[0]
q1f_as_q3 = movies['audience_score'].quantile([0.75]).iloc[0]
q1f_as_p83 = movies['audience_score'].quantile([0.83]).iloc[0]

# # For imdb_rating-
q1f_ir_q1 =movies['imdb_rating'].quantile([0.25]).iloc[0]
q1f_ir_p37 = movies['imdb_rating'].quantile([0.37]).iloc[0]
q1f_ir_q3 = movies['imdb_rating'].quantile([0.75]).iloc[0]
q1f_ir_p83 = movies['imdb_rating'].quantile([0.83]).iloc[0]
```

In [17]:
```
grader.check("q1f")
```

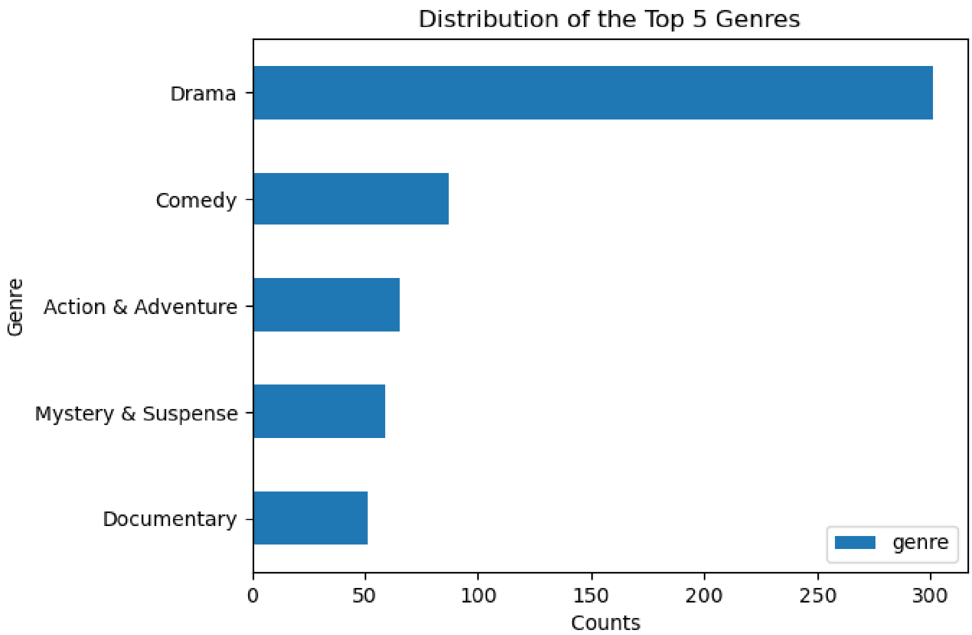
Out[17]: `q1f` passed! 🍀

Q1(g) Visualization: Single Variable

I highly encourage looking at the [Fundamentals of Visualization (https://clauswilke.com/dataviz/index.html)](https://clauswilke.com/dataviz/index.html) reference book to guide in the creation of "good" visualizations requested below.

Create a bar plot for the `genre` variable; display only the the top 5 genres.

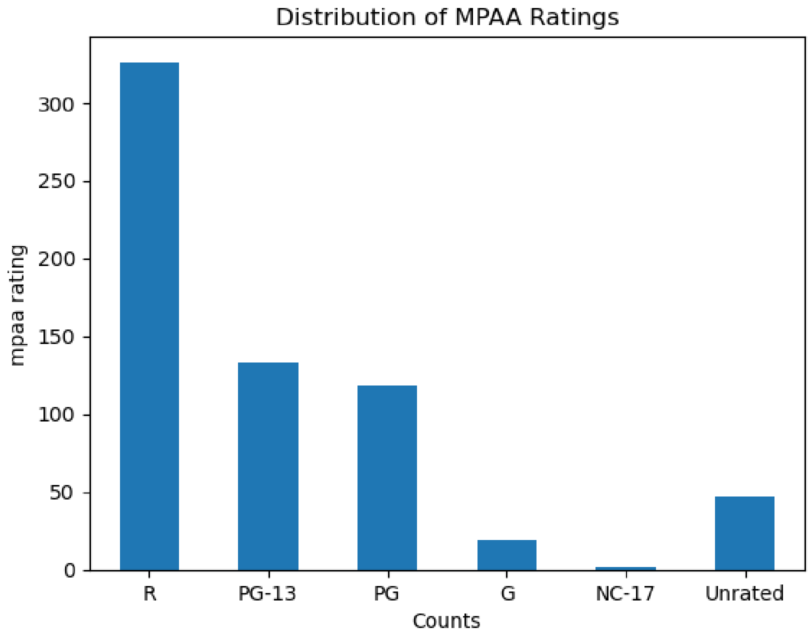
In [18]:
```
# Create bar plot for "genre", with top 5 genres

movies['genre'].value_counts(ascending=False).to_frame()[:5].sort_values(by='genre').plot(kind='barh')
plt.title("Distribution of the Top 5 Genres")
plt.xlabel("Counts ")
plt.ylabel("Genre")
plt.show()
```



Q1(h) Visualization: Single Variable

Create a horizontal bar plot for `mpaa_rating`, sorted by rating (let 'Unrated' be last).

In [19]:
```
# Create a horizontal bar plot for mpaa_rating, sorted by rating.
movies_mpaa=movies['mpaa_rating'].value_counts(ascending=False)
Unrated_val=movies_mpaa['Unrated']
movies_mpaa.drop('Unrated',inplace=True)
movies_mpaa['Unrated']=Unrated_val
movies_mpaa.plot(kind='bar')
plt.title("Distribution of MPAA Ratings")
plt.xlabel("Counts")
plt.ylabel("mpaa rating")
plt.xticks(rotation=0)
plt.show()
```

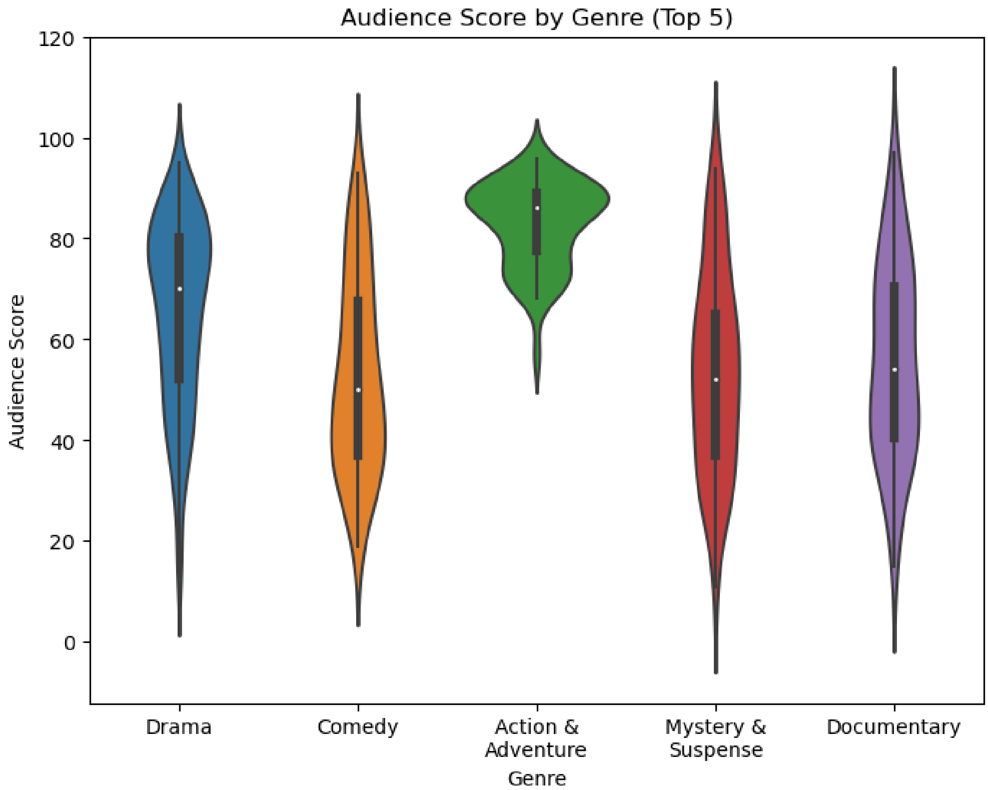


Q1(i) Visualization: Two Variables

Create a violin plot for `audience_score` grouped by `genre` (use only the top 5 genres).

You may want to loop at the `textwrap` module to get ticklabels formatted well.

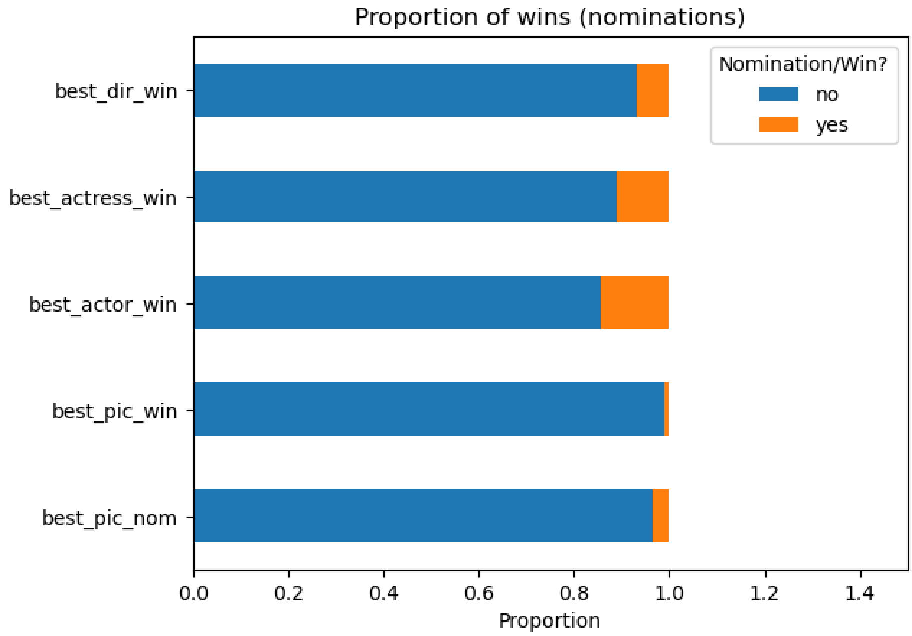
In [46]:
```
# Create a violin plot for `audience_score` grouped by top 5 `genres`
# Get top 5 genres via groupby method
top_5_genres = movies.groupby(by='genre')['audience_score'].count().sort_values(ascending=False).iloc[:5].index
 # Filter movies by top 5 genres
movies_top_5 = movies[movies['genre'].isin(top_5_genres)]
fig=plt.figure(figsize=(8,6))
fig = sns.violinplot(x='genre', y='audience_score', data=movies_top_5)
fig.set_xlabel('Genre')
fig.set_ylabel('Audience Score')
fig.set_title('Audience Score by Genre (Top 5)')

labels = [ '\n'.join(textwrap.wrap(tl, 15)) for tl in top_5_genres] # Format tick labels by loop at textwrap
fig.set_xticklabels(labels)

plt.show()
```



Q1(j) Visualization: Multiple variables

Create a stacked bar chart to display the proportion of wins (nominations) for the 5 `best_*` variables.

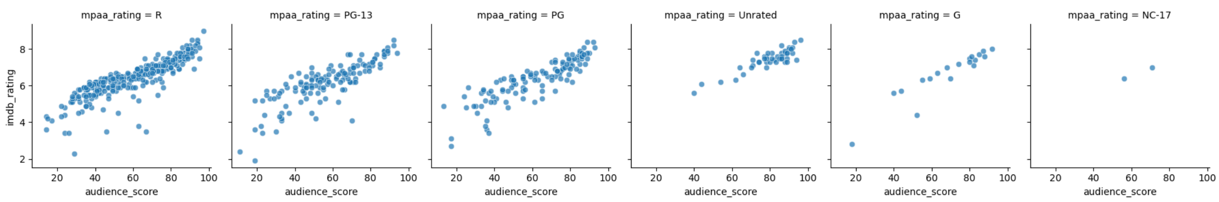
In [47]:
```
# Create a stacked bar chart to display the proportion of wins (nominations)
#   for the 5 `best_*` variables.
best_col_names=list(filter(lambda text: 'best_' in text, movies.columns)) #filtering out the columns containing
best_cols=movies[best_col_names]
proportions = best_cols.apply(lambda x: pd.value_counts(x, normalize=True)).T #Find proportions of yes and no

# Create stacked bar chart
fig = proportions.plot(kind='barh', stacked=True)
fig.set_xlabel('Proportion')
fig.legend(title='Nomination/Win?', loc='upper right')
fig.set_title('Proportion of wins (nominations)')
fig.set_xlim(0,1.5)
plt.show()
```



Q1(k) Visualization: Multiple variables

Create a small multiples (or faceted) scatter plot of `imdb_rating` vs. `audience_score` for each `mpaa_rating` .

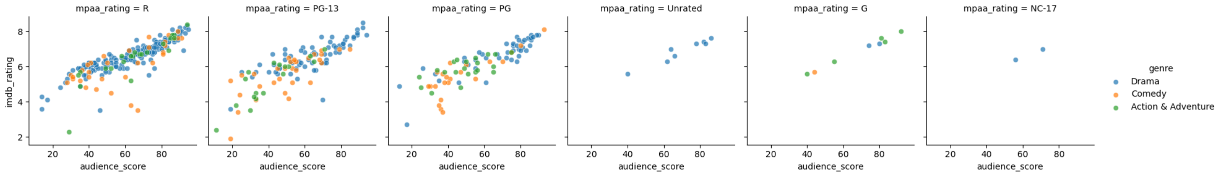
In [48]:
```
# Create a small multiples (or faceted) scatter plot of `imdb_rating` vs.
#   `audience_score` for each `mpaa_rating`.
g = sns.FacetGrid(movies, col="mpaa_rating")
g.map(sns.scatterplot,"audience_score","imdb_rating", alpha=.7)
plt.show()
```



Q1(bonus)

Create a small multiples (or faceted) scatter plot of `imdb_rating` vs. `audience_score` for each `mpaa_rating` , colored with the top 3 `genres` .

In [49]:
```python
# Create a small multiples (or faceted) scatter plot of `imdb_rating` vs.
#   `audience_score` for each `mpaa_rating`, colored by the top 3 `genre`

top_3_genres = movies['genre'].value_counts(ascending=False)[:3].index# Get top 3 genres via groupby method
movies_top_3= movies[movies['genre'].isin(top_3_genres)] # Filter movies by top 3 genres

g = sns.FacetGrid(movies_top_3, col="mpaa_rating", hue="genre")
g.map(sns.scatterplot,"audience_score","imdb_rating", alpha=.7)
g.add_legend()
plt.show()
```



Q2 - Performance Metrics

Write a function to calculate: (a) true positive rate, (b) false positive rate, (c) accuracy, and (d) Matthews Correlation Coefficient (MCC).

You can make use of `sklearn.metrics` functions.

The function will have inputs of `y_true` (np.array) - the true label for a set of samples and `y_pred` (np.array) - the predicted labels for a set of samples, and a threshold `thres_value` (float).

The function returns a list of the true positive rate, false positive rate, accuracy and MCC for the inputs where the predicted labels are thresholded at the provided value (using >= comparisons).

This function will then be used to create a DataFrames with rows corresponding with the 10 thresholds (y_pred values) and columns reporting the different thresholds, the true positive rate (TPR), false positive rate (FPR), accuracy (ACC), and Matthews correlation coefficient (MCC).

In [50]:

```python
def calc_metrics(y_true, y_pred, thres_value):
    # Calculate tpr, fpr, accuracy, and MCC on input
    # Input:
    #  y_true - sample labels       (np.array)
    #  y_pred - sample predictions  (np.array)
    #  thres_value - threshold for predictions,  >=
    # Return list of tpr, fpr, accuracy, and MCC


    y_pred_binary = (y_pred >= thres_value).astype(int)
    TN, FP, FN, TP = confusion_matrix(y_true, y_pred_binary).ravel()

    tpr_val = TP/(TP+FN)
    fpr_val = FP/(FP+TN)
    acc_val = (TP+TN)/(TP+FP+FN+TN)
    mcc_val = (TP*TN - FP*FN) / ((TP+FP)*(TP+FN)*(TN+FP)*(TN+FN))**(1/2)
    mcc_val = np.nan_to_num(mcc_val, copy=True, nan=0.0)


    return tpr_val, fpr_val, acc_val, mcc_val


y_true = np.array([1,1,0,1,1,0,1,0,0,0])
y_pred = np.array([0.98,0.92,0.85,0.77,0.71,0.64,0.57,0.42,0.34,0.32])


perfDF = pd.DataFrame(columns = ['Threshold', 'TPR', 'FPR', 'ACC', 'MCC'])
for thres in y_pred:
    tpr_val, fpr_val, acc_val, mcc_val = calc_metrics(y_true, y_pred, thres)
    perfDF = perfDF.append({'Threshold' : thres, 'TPR' : tpr_val,
                            'FPR' : fpr_val, 'ACC': acc_val, 'MCC': mcc_val},
                           ignore_index = True)

perfDF
```

```
/tmp/ipykernel_594534/3010649057.py:16: RuntimeWarning: invalid value encountered in true_divide
  mcc_val = (TP*TN - FP*FN) / ((TP+FP)*(TP+FN)*(TN+FP)*(TN+FN))**(1/2)
```

Out[50]:

	Threshold	TPR	FPR	ACC	MCC
0	0.98	0.2	0.0	0.6	0.333333
1	0.92	0.4	0.0	0.7	0.500000
2	0.85	0.4	0.2	0.6	0.218218
3	0.77	0.6	0.2	0.7	0.408248
4	0.71	0.8	0.2	0.8	0.600000
5	0.64	0.8	0.4	0.7	0.408248
6	0.57	1.0	0.4	0.8	0.654654
7	0.42	1.0	0.6	0.7	0.500000
8	0.34	1.0	0.8	0.6	0.333333
9	0.32	1.0	1.0	0.5	0.000000

In [51]:

```python
grader.check("q2")
```

Out[51]: **q2** passed! 🎉

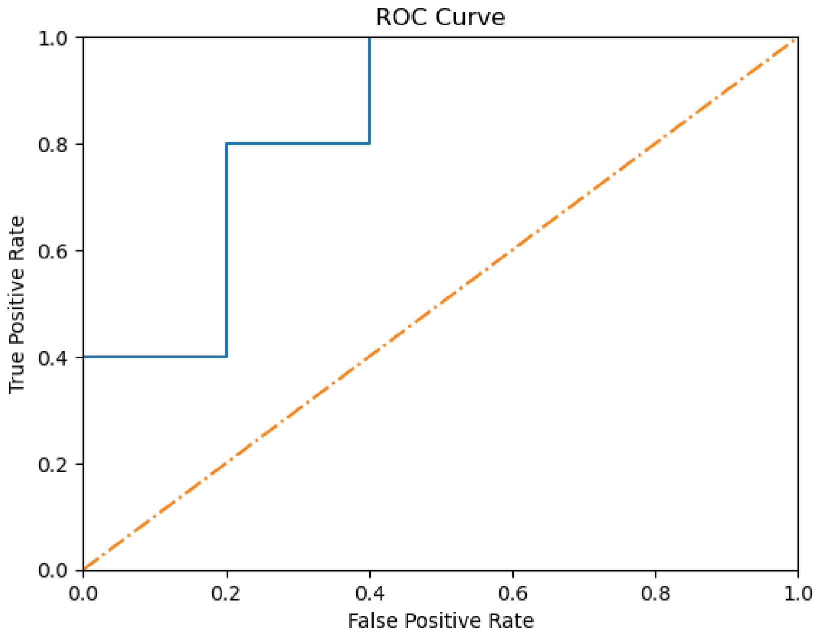
Q3 - Plot ROC Curve:

Use the results from Question 2 to plot the ROC curve for the data.

Note, plot this curve using the standard plotting tools rather than any special library/package available for making ROC plots.

In [26]:

```
# Create a ROC curve using the results from Q2
plt.plot(perfDF['FPR'], perfDF['TPR'])
plt.plot([0, 1], [0, 1], linestyle='-.')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()
```



Q4 - NBA Winners

For this problem you will use a data set of NBA basketball games (2016 - 2021 seasons). The dataset was collected from the NBA website API - [https://www.nba.com (https://www.nba.com)](https://www.nba.com)

You will use the data to predict whether team A in each matchup will win the game.

The data consists of variables:

- `SEASON_ID` , `GAME_ID` , `GAME_DATE` - variables to identify individual samples (ignore for prediction)
- `TEAM_A` , `TEAM_B` , `MATCHUP` - variables describing the two teams in a game (ignore for prediction)
- `WON` - This is the target / class feature to be predicted

The remaining variables are predictor variables for the models. They come in pairs "*_DIFF" and "*_A" reporting the given statistic as the difference between Team A and Team B and the statistic itself for Team A.

- `FG_PCT_DIFF` , `FG_PCT_A` - field goal percentage.
- `FG3_PCT_DIFF` , `FG3_PCT_A` - percentage of 3-point shots made.
- `FT_PCT_DIFF` , `FT_PCT_A` - percentage of free throws made.
- `REB_DIFF` , `REB_A` - number of rebounds.
- `AST_DIFF` , `AST_A` - number of assists.
- `STL_DIFF` , `STL_A` - number of steals.
- `TOV_DIFF` , `TOV_A` - number of turnovers.
- `PF_DIFF` , `PF_A` - number of personal fouls.

Q4(a) - Load Data

Load the `nba` data and drop the samples with missing data.

Create a DataFrame `nbaX` with the predictor variables and `nbaY` with the target variable.

In [52]:
```python
na_values = [" ", '?', ' ?', "NA", "N/A", "NULL","nan", "NaN"]
nba = pd.read_csv("data/nba-simple.csv", na_values=na_values)
nba = nba.dropna()

nbaX = nba.drop(['WON','SEASON_ID','GAME_ID','GAME_DATE','TEAM_A','TEAM_B','MATCHUP'],axis=1)
nbaY = nba['WON'].copy()

nba.head()
```

Out[52]:

	SEASON_ID	GAME_ID	GAME_DATE	TEAM_A	TEAM_B	MATCHUP	WON	FG_PCT_DIFF	FG_PCT_A	FG3_PCT_DIFF	...	REB_DIFF
0	42019	41900406	10/11/20	MIA	LAL	MIA vs. LAL	0	-0.040	0.443	0.043	...	-5
1	42019	41900405	10/9/20	LAL	MIA	LAL vs. MIA	0	0.005	0.463	-0.056	...	6
2	42019	41900404	10/6/20	MIA	LAL	MIA vs. LAL	0	-0.016	0.427	-0.015	...	-3
3	42019	41900403	10/4/20	MIA	LAL	MIA vs. LAL	1	0.083	0.513	0.020	...	-6
4	42019	41900402	10/2/20	LAL	MIA	LAL vs. MIA	1	-0.002	0.505	-0.067	...	7

5 rows × 23 columns

In [53]:
```python
grader.check("q4a")
```

Out[53]: **q4a** passed! 🎉

Q4(b) - Labels

Let's understand the what we should expect as a baseline performance for predicting whether team A wins.

(i). What fraction of games has team A as the winner? Value should be in between 0 and 1.
(ii). What should a constant classifier model predict? A *constant classifier* always predicts the same value no matter the input.
(iii). What is the error rate of the constant classifier? Value should be in between 0 and 1.

Answer the following questions below. Note, you should not use any `sklearn` functions, but simply look at properties of the data labels.

In [54]:
```python
q4b_i = nbaY.mean() #Since the target values contains binary values (0 or 1),
                    #the mean gives us the proportion of 1's in the column,
                    #which is equivalent to the fraction of games won by team A.

q4b_ii = nbaY.mode()[0] #The most common outcome in the target values is the value
                        #that a constant classifier should predict (i.e. 1).

q4b_iii = (nbaY != q4b_ii).mean() #The error rate is the mean of the records
                                  #where value is not equal to the most common outcome.
```

In [55]:
```python
grader.check("q4b")
```

Out[55]: **q4b** passed! 🚀

Q4(c) Model Selection and Evaluation: Three-fold Split

Split the data into training, validation and test sets with 60, 20, and 20% of the data respectively. Make sure to split the data such that the distribution of class labels is approximately equal across splits - “stratify”.

Set the seed for the random generator in `random_state` to ”4821”

In [56]:
```python
# Split of the test set
X_trainval, X_test, y_trainval, y_test = train_test_split(
    nbaX, nbaY, test_size=0.20, random_state=4821, stratify=nbaY)

# Split trainval into train + val
X_train, X_val, y_train, y_val = train_test_split(
    X_trainval, y_trainval, test_size=0.25, random_state=4821, stratify=y_trainval)
```

In [57]:
```
grader.check("q4c")
```
Out[57]: **q4c** passed! 🚀

Q4(d) Scaling

Scale the predictor data with standard scaling or normalizaiton.

Make sure to use training data set to set scaling parameters and apply those parameters to scaling the validation and testing data.

Create scaled train+val data on this same set to build the best model, and use those parameters to scale the test data to evaluate the best model.

Helpful functions: Python - `StandardScaler` from `sklearn.preprocessing` .

In [58]:
```
scaler = StandardScaler().fit(X_train)
X_train_sc = scaler.transform(X_train)
X_val_sc = scaler.transform(X_val)

scaler_final = StandardScaler().fit(X_trainval)
X_trainval_sc = scaler_final.transform(X_trainval)
X_test_sc = scaler_final.transform(X_test)
```

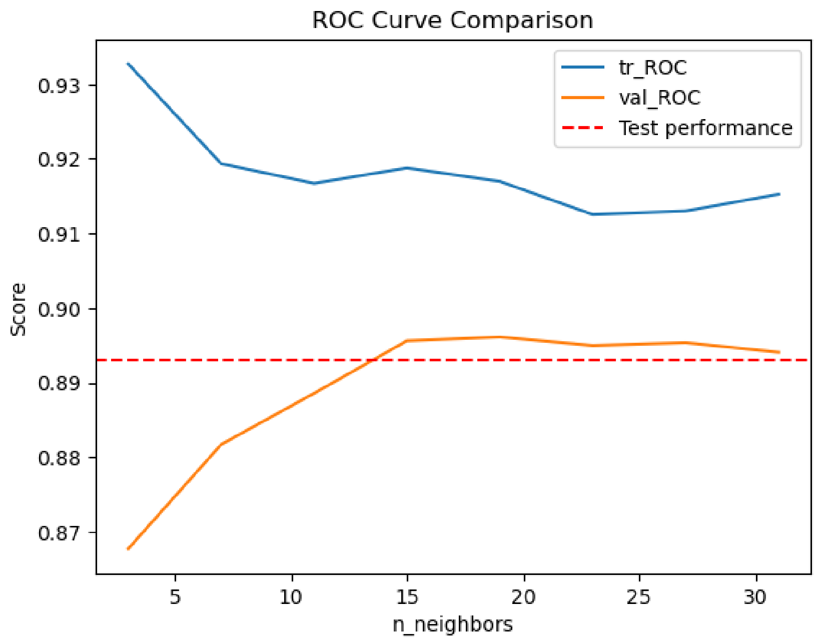
In [59]:
```
grader.check("q4d")
```
Out[59]: **q4d** passed! 💯

Q4(e) KNN - K Nearest Neighbors

For values of k, [3, 7, 11, 15, 19, 23, 27, 31], fit a k-nearest-neighbor classifier to the training data.

- fit a k nearest neighbors model to the training data
- evaluate the classifier on the training and validation set using auc
- select the best value of k
- create a best model on train+validation
- evaluate the classifier on the test data.
- plot the training and validation performance vs k;
  add a line showing the test performance

In [60]:
```python
kvals = [3, 7, 11, 15, 19, 23, 27, 31]

knn_auc_val = np.zeros((1,len(kvals)))
knn_auc_tr = np.zeros((1,len(kvals)))


# fit a k nearest neighbors model to the training data
for n in kvals:
    # build a model
    knn = KNeighborsClassifier(n_neighbors=n)
    knn.fit(X_train_sc, y_train)

    # evaluate the classifier on the training and validation set using auc
    y_pred_train  = knn.predict(X_train_sc)
    knn_auc_tr[0][kvals.index(n)] = metrics.roc_auc_score(y_train, y_pred_train)

    y_pred_val  = knn.predict(X_val_sc)
    knn_auc_val[0][kvals.index(n)] = metrics.roc_auc_score(y_val, y_pred_val)


# select the best value of k
knn_bestk = kvals[np.argmax(knn_auc_val)]


# # create a best model on train+validation
knn = KNeighborsClassifier(n_neighbors=knn_bestk)
knn.fit(X_trainval_sc, y_trainval)


# # evaluate the classifier on the test data.
y_pred_test  = knn.predict(X_test_sc)
knn_auc_test = metrics.roc_auc_score(y_test, y_pred_test)

# # plot the training and validationing performance vs k.
# #   add a line for test performance

plt.plot(kvals, knn_auc_tr[0], label='tr_ROC')
plt.plot(kvals, knn_auc_val[0], label='val_ROC')
# plt.plot(knn_bestk, knn_auc_test)
plt.xlabel('n_neighbors')
plt.ylabel('Score')
plt.title('ROC Curve Comparison')
plt.axhline(y=knn_auc_test, color='r', linestyle='--', label='Test performance')
plt.legend()
plt.show()
print('Best k:  %d' % (knn_bestk))
print('Test Perf:  %.6f' % (knn_auc_test))
```




```
Best k:  19
Test Perf:  0.893046
```

In [61]:
```python
grader.check("q4e")
```

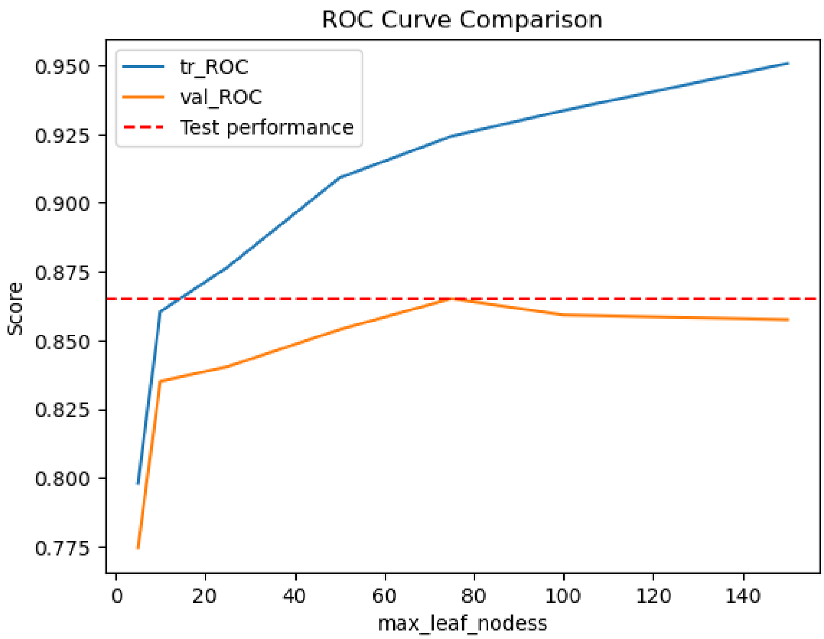
Out[61]: **q4e** passed! ✨

Q4(f) Decision Trees

For values of max_leaf_nodes nodes [5, 10, 25, 50, 75, 100, 150], fit the Decision Trees classifier to the training data.

- fit a decision tree model to the training data (use `random_state=4821`)
- evaluate the classifier on the training and validation set using auc
- select the best max_leaf_nodes
- retrain the best model on train+validation
- report the auc on the testing data.
- plot the train and validation auc vs max_leaf_nodes;
  add a line for the test auc performance
- print out the best tree.

In [62]:
```
nodes = [5, 10, 25, 50, 75, 100, 150]

dt_auc_val = np.zeros((1,len(nodes)))
dt_auc_tr = np.zeros((1,len(nodes)))

# fit a decision tree model to the training data (use random_state=4821)
for n in nodes:
    # build a model
    dt = tree.DecisionTreeClassifier(max_leaf_nodes=n, random_state=4821)
    dt.fit(X_train_sc, y_train)

# evaluate the classifier on the training and validation set using auc
    y_pred_train  = dt.predict(X_train_sc)
    dt_auc_tr[0][nodes.index(n)] = metrics.roc_auc_score(y_train, y_pred_train)

    y_pred_val  = dt.predict(X_val_sc)
    dt_auc_val[0][nodes.index(n)] = metrics.roc_auc_score(y_val, y_pred_val)

# select the best max_leaf_nodes
dt_bestn = nodes[np.argmax(dt_auc_val)]

# retrain the best model on train+validation
dt = tree.DecisionTreeClassifier(max_leaf_nodes=dt_bestn, random_state=4821)
dt.fit(X_trainval_sc, y_trainval)

# report the auc on the testing data.
y_pred_test  = dt.predict(X_test_sc)
dt_auc_test = metrics.roc_auc_score(y_test, y_pred_test)

# plot the train and validation auc vs max_leaf_nodes.
#   add a line for test performance
plt.plot(nodes, dt_auc_tr[0], label='tr_ROC')
plt.plot(nodes, dt_auc_val[0], label='val_ROC')
# plt.plot(dt_bestn, dt_auc_test)
plt.xlabel('max_leaf_nodess')
plt.ylabel('Score')
plt.title('ROC Curve Comparison')
plt.axhline(y=dt_auc_test, color='r', linestyle='--', label='Test performance')
plt.legend()
plt.show()

print('Best max_leaf_nodes:  %d' % (dt_bestn))
print('Test Perf:  %.6f' % (dt_auc_test))
```

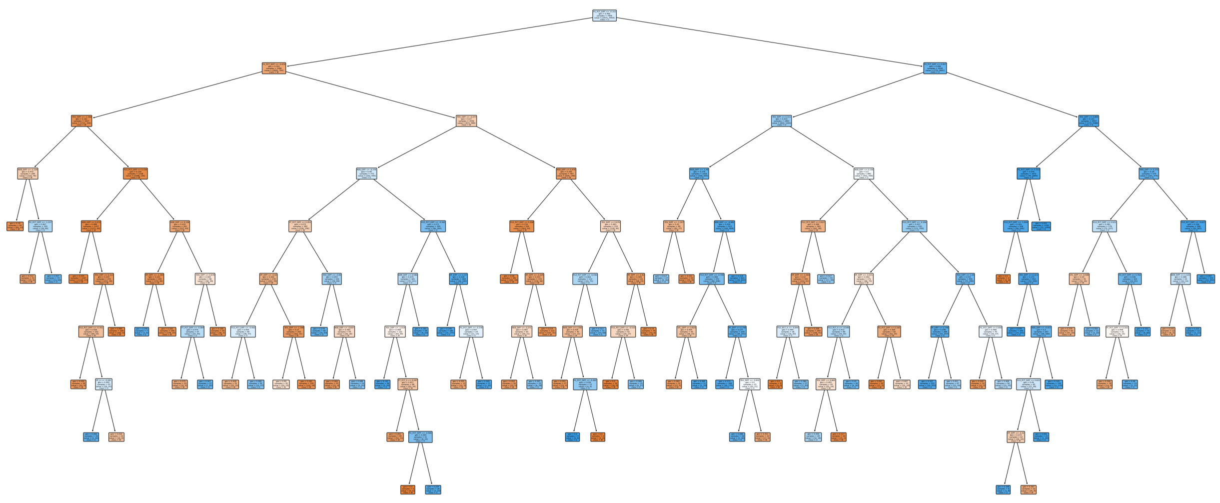



```
Best max_leaf_nodes:  75
Test Perf:  0.865113
```

In [63]:
```
# print out the tree

fig = plt.figure(figsize=(46,20))
_ = tree.plot_tree(dt,
                   feature_names=nbaX.columns,
                   class_names='WON',
                   rounded=True,
                   filled=True)
plt.title('The Best Tree',fontdict= {'fontsize': 50})
```

Out[63]: Text(0.5, 1.0, 'The Best Tree')

The Best Tree



In [64]:
```
grader.check("q4f")
```

Out[64]: **q4f** passed! ✨

Q4(g) Naive Bayes

Use the GaussianNB on training + validation data and report the training+val and testing data performance (auc).

In [65]:
```
# q4g

gnb = naive_bayes.GaussianNB()
gnb.fit(X_trainval_sc, y_trainval)

y_pred_trainval = gnb.predict(X_trainval_sc)
y_pred_test = gnb.predict(X_test_sc)

nb_auc_trainval = metrics.roc_auc_score(y_trainval, y_pred_trainval)
nb_auc_test = metrics.roc_auc_score(y_test, y_pred_test)
print('Training+Val Perf:  %.6f' % (nb_auc_trainval))
print('Test Perf:  %.6f' % (nb_auc_test))
```

```
Training+Val Perf:  0.859176
Test Perf:  0.842170
```

In [66]:
```
grader.check("q4g")
```

Out[66]: **q4g** passed! 🙌

Q4(h) Support Vector Machines + GridSearch with Cross-validation (without using GridSearchCV)

In this part, you will use the *do-it-yourself* approach using `StratifiedKFold` (rather than GridSearchCV).

Use the same split from above with 80% train+val, 20% test data.

With the train+val data, use 10-fold cross-validation (make sure to use Stratified approach with random_state = 4821). Train each model on the training set and evaluate each model on the validation set. Consider how to do scaling with this approach. You will consider SVM models with the following hyperparameters:

- Polynomial kernel with C = [10^-2, 10^-1, 1], degree = [1, 2, 3]
- RBF kernel (Gaussian kernel) with C = [10^-2, 10^-1, 1]

Collect each model's validation performance.

Report the mean validation performance (AUC) as DataFrame with:

- rows, Linear kernel, poly kernel d=2, poly kernel d=3, rbf kernel
- columns, C = [10^-2, 10^-1, 1]

Report the best parameter combination (cost + kernel).

Retrain the best model on train+val (same used above for the other classifiers) and report the test performance.

In [67]:
```python
# With the train+val data, use 10-fold cross-validation (with StratifiedKFold )
cv = StratifiedKFold(n_splits=10, shuffle=True,random_state=4821)

# Train each model on the train set, evaluate each model on the validation set
#  set up scaling

scaler = StandardScaler()

# You will consider SVM models with the following hyperparameters:
#   - Polynomial kernel with C = [10^-2, 10^-1, 1], degree = [1, 2, 3]
#   - RBF kernel (Gaussian kernel) with C = [10^-2, 10^-1, 1]
# Define the hyperparameter combinations to try

kernels=['Linear kernel', 'poly kernel d=2', 'poly kernel d=3', 'rbf kernel']
C = [10**-2, 10**-1, 1]

#Dictionaries to use later to obtain the best combination of parameters
C_dict={10**-2:'10^-2',10**-1:'10^-1', 1:'1'}
C_dict_reverse={'10^-2':10**-2,'10^-1':10**-1, '1':1}
kernel_dict={'Linear kernel':'linear', 'poly kernel d=2':'poly', 'poly kernel d=3':'poly', 'rbf kernel':'poly'}
Degree_dict={'Linear kernel':3, 'poly kernel d=2':2, 'poly kernel d=3':3, 'rbf kernel':'NA'}


# Collect each model's validation performance.

# Report the mean validation performance (AUC) as DataFrame with:
#   - rows, Linear kernel, poly kernel d=2, poly kernel d=3, rbf kernel
#   - columns, C = [10^-2, 10^-1, 1]

svm_results = pd.DataFrame(columns=['10^-2', '10^-1', '1'],
                           index=['Linear kernel', 'poly kernel d=2', 'poly kernel d=3', 'rbf kernel' ])

for kernel in kernels:
    for c in C:
        if kernel=='Linear kernel':
            model=svm.SVC(kernel='poly', C=c, degree=1,coef0=1)
        elif kernel == 'poly kernel d=2':
            model=svm.SVC(kernel='poly', C=c, degree=2,coef0=1)
        elif kernel == 'poly kernel d=3':
            model=svm.SVC(kernel='poly', C=c, degree=3,coef0=1)
        elif kernel == 'rbf kernel':
            model=svm.SVC(kernel='rbf', C=c)

        # Initialize the running sum of validation AUC scores
        val_auc_sum = 0

        # Iterate over the 10 folds of cross-validation
        for train_idx, val_idx in cv.split(X_trainval, y_trainval):
            # Split the data into train and validation sets
            X_tr, y_tr = X_trainval.iloc[train_idx], y_trainval.iloc[train_idx]
            X_val, y_val = X_trainval.iloc[val_idx], y_trainval.iloc[val_idx]

            # Scale the data
            X_tr = scaler.fit_transform(X_tr)
            X_val = scaler.transform(X_val)

            # Train the model on the train set
            model.fit(X_tr, y_tr)

            # Evaluate the model on the validation set
            y_pred_val = model.predict(X_val)
            val_auc = roc_auc_score(y_val, y_pred_val)

            # Add the validation AUC score to the running sum
            val_auc_sum += val_auc

        # Calculate the mean validation AUC score for this hyperparameter combination
        mean_val_auc = val_auc_sum / 10

        svm_results[C_dict[c]].loc[kernel]=mean_val_auc

# # Report the best parameter combination (cost + kernel).
max_value = np.max(svm_results.values) # Get the maximum value
max_index = np.where(svm_results.values == max_value) # Get the index of the maximum value
max_row, max_col = max_index[0][0], max_index[1][0] # Get the row and column of the maximum value
svm_bestC = C_dict_reverse[svm_results.columns[max_col]]
svm_bestKernel =kernel_dict[svm_results.reset_index().iloc[max_row]['index']]
svm_bestD = Degree_dict[svm_results.reset_index().iloc[max_row]['index']]

# Retrain the best model on train+val (same data used above for the other
# classifiers) and report the test performance .
best_model=model=svm.SVC(kernel=svm_bestKernel, C=svm_bestC, degree=svm_bestD, coef0=1)
X_trainval_sc=scaler.fit_transform(X_trainval)
X_test_sc= scaler.transform(X_test)
best_model.fit(X_trainval_sc,y_trainval)
```

```
y_pred_test = best_model.predict(X_test_sc)
svm_auc_test = metrics.roc_auc_score(y_test, y_pred_test)
print('Test Perf:  %.6f' % (svm_auc_test))

svm_results
```

```
Test Perf:  0.928970
```

Out[67]:

	10^-2	10^-1	1
Linear kernel	0.914331	0.924569	0.927308
poly kernel d=2	0.920994	0.928482	0.929375
poly kernel d=3	0.922563	0.929508	0.928288
rbf kernel	0.890947	0.922462	0.928259

In [68]:
```
grader.check("q4h")
```

Out[68]: **q4h** passed! 💯

Q4(i) Ensemble Methods + GridSearchCV with Pipelines

Let's examine bagging & boosting ensemble approaches for prediction.

For this part, we will use the ultimately preferred method for building predictor models by using pipelines.

You will create a pipeline for both the Random Forest models and the AdaBoost models. Both pipelines will use standard scaling preprocessing.

For the random forest, consider hyper-parameters for the maximum number of features: [2, 4, 8, 16] and number of estimators of [25, 50, 100]. For AdaBoost, consider the hyper-parameter of the number of estimators as [10, 25, 50, 100].

To ensure repeatability or your code (and to compare to the autograder) make sure to set the random state in both classifiers and the stratified 10-fold cross-validation to "4821".

Use AUC as the scoring metric for the GridSearch criteria.

You will need to report the best hyper-parameters for both models as well as the final test set performance.

In [69]:

```
# You will create a pipeline for both the Random Forest and AdaBoost models.
#   Both pipelines will use standard scaling preprocessing.
ab_pipe = Pipeline([("scaler", StandardScaler()),("ab", AdaBoostClassifier(random_state=4821))])

rf_pipe = Pipeline([("scaler", StandardScaler()),("rf", RandomForestClassifier(random_state=4821))])

# RF: hyper-parameters for the maximum number of features: [2, 4, 8, 16] and
#   number of estimators of [25, 50, 100].
rf_params = {"rf__n_estimators": [25, 50, 100],"rf__max_features": [2, 4, 8, 16]}

# AdaBoost: hyper-parameter of the number of estimators as [10, 25, 50, 100].
ab_params = {"ab__n_estimators": [10, 25, 50, 100]}

# Set the random state in both classifiers and the stratified k-fold cv to 4821
cvStrat = StratifiedKFold(n_splits=10, shuffle=True, random_state=4821)

# Use AUC as the scoring metric for the GridSearch criteria.
rf_grid = GridSearchCV(rf_pipe, rf_params, cv=cvStrat, scoring="roc_auc")
rf_grid.fit(X_trainval, y_trainval)

ab_grid = GridSearchCV(ab_pipe, ab_params, cv=cvStrat, scoring="roc_auc")
ab_grid.fit(X_trainval, y_trainval)

# Report the best hyper-parameters and final test set performance
rf_best_params = rf_grid.best_params_
ab_best_params = ab_grid.best_params_


y_pred_test=rf_grid.predict(X_test)
rf_auc_test = metrics.roc_auc_score(y_test, y_pred_test)
print('Random Forest Test Perf:  %.6f' % (rf_auc_test))

y_pred_test=ab_grid.predict(X_test)
ab_auc_test = metrics.roc_auc_score(y_test, y_pred_test)
print('AdaBoost Test Perf:  %.6f' % (ab_auc_test))
```

```
Random Forest Test Perf:  0.906677
AdaBoost Test Perf:  0.916798
```

In [70]:

```
grader.check("q4i")
```

Out[70]: **q4i** passed! ✨

Submission

1. Make sure you have run all cells in your notebook in order, so that all images/graphs appear in the output, then save your notebook.
2. Print your notebook to a PDF
   - Using Jupyter Lab
     - Option A: Select "File" -> "Print ..." save as a PDF In my test on my local machine with Safari and Chrome on MacOS, the resulting PDF is saved to the Downloads folder on your local machine.
     - Option B: Select "File" -> "Save and Export Notebook as ..." -> HTML, then open HTML file in a Browser and Print to PDF
       In my test, the resulting HTML file is saved to the Downloads folder. Then, can print the html to a PDF using the browser.
   - Using Jupyter notebook
     - Select "File" -> "Download as" -> "HTML", then open HTML file in a Browser and Print to PDF
       In my test, the resulting HTML file is saved to the Downloads folder. Then, can print the html to a PDF using the browser.
3. Gather the PDF and HTML together.
4. Zip the notebook and PDF together and submit on Gradescope

In [ ]: