Game Classification Model

execute in same directory as 'game_data.csv'

In [1]:
```
import os, math
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

# needs to be reset if a model is saved
# os.chdir(working_dir_path)
working_dir = os.getcwd()
game_data = pd.read_csv('game_data.csv')
game_data.head() # See the first 5 rows to check data import
```

Out[1]:

	id	type	name	year	minplayers	maxplayers	playingtime	minplaytime	maxplayti
0	13	boardgame	Catan	1995	3	4	120	60	1
1	822	boardgame	Carcassonne	2000	2	5	45	30	
2	30549	boardgame	Pandemic	2008	2	4	45	45	
3	68448	boardgame	7 Wonders	2010	2	7	30	30	
4	36218	boardgame	Dominion	2008	2	4	30	30	

5 rows × 22 columns

clean and filter data

selective filtering for removing extremes, retain 'game_data' in case any reference to complete set is needed

In [2]:
```
bgg_games = game_data[game_data['maxplayers'] <= 30]
```

```
bgg_games = bgg_games[bgg_games['minplaytime'] <= 180]
bgg_games = bgg_games[bgg_games['maxplaytime'] <= 720]
bgg_games = bgg_games[bgg_games['minage'] <= 21]
bgg_games = bgg_games[bgg_games['playingtime'] >= 10]
bgg_games = bgg_games[bgg_games['maxplayers'] >= bgg_games['minplayers']]
```

first selection of columns potentially relevant to 'categories'

In [3]:
```
dtc_test = bgg_games[['type', 'year', 'minplayers', 'maxplayers', 'playingtime',
       'minplaytime', 'maxplaytime', 'minage', 'users_rated', 'avg_rating',
       'bay_rating', 'owners', 'traders', 'wanters', 'wishers',
       'total_comments', 'total_weights', 'complexity', 'categories',
       'mechanics']]
```

convert strings of multiple values into lists

In [4]:
```
dtc_test['categories'] = dtc_test['categories'].apply(lambda x: x.strip(']['').split(',
dtc_test['mechanics'] = dtc_test['mechanics'].apply(lambda x: x.strip(']['').split(', ')
```

count number of mechanics and categories for each game, make new columns

In [5]:
```
dtc_test['num_mechs'] = dtc_test.apply(lambda row: len(row['mechanics']), axis=1)
dtc_test['num_cats'] = dtc_test.apply(lambda row: len(row['categories']), axis=1)
dtc_test['rating_diff'] = dtc_test.avg_rating - dtc_test.bay_rating
dtc_test['player_diff'] = dtc_test.maxplayers - dtc_test.minplayers
```

In [6]:
```
def split_data_frame_list(df, target_column, output_type=str):
    '''
    Accepts a column with list values and splits into several rows.

    df: dataframe to split
    target_column: the column containing the values to split
    output_type: type of all outputs
    returns: a dataframe with each entry for the target column separated, with each ele
    The values in the other columns are duplicated across the newly divided rows.
    '''
    row_accumulator = []

    def split_list_to_rows(row):
        split_row = row[target_column]
        if isinstance(split_row, list):
          for s in split_row:
              new_row = row.to_dict()
              new_row[target_column] = output_type(s)
              row_accumulator.append(new_row)
        else:
          new_row = row.to_dict()
          new_row[target_column] = output_type(split_row)
          row_accumulator.append(new_row)

    df.apply(split_list_to_rows, axis=1)
    new_df = pd.DataFrame(row_accumulator)

    return new_df
```

split 'mechanics' lists into multiple rows for decision tree training

In [7]:
```
dtc_test = split_data_frame_list(dtc_test, 'mechanics')
```

Before filtering by year for model, split and test to remove old mechanics/categories that are no longer relevant

In [8]:
```
# year by which to remove uniques to prevent models from being incompatible
# -math.inf will include all games in dataframe
break_year = 2000 #-math.inf

old_mechs = list(dtc_test[dtc_test['year'] < break_year].mechanics.unique())
new_mechs = list(dtc_test[dtc_test['year'] >= break_year].mechanics.unique())
unique_old_mechs = list(set(old_mechs).difference(new_mechs))

# remove rows with irrelevant mechanics
dtc_test = dtc_test[~dtc_test.mechanics.isin(unique_old_mechs)]
```

second selection of columns to further filter for decision tree model

In [9]:
```
# columns to use
desired_cols = ['type', 'year', 'minplayers','maxplayers', 'playingtime', 'minplaytime'

# columns available
#all_cols = ['type', 'year', 'minplayers', 'maxplayers', 'playingtime', 'minplaytime',

# make dummies and attach to frame for tree model, leave categories alone
total_frame = dtc_test[desired_cols]
mech_dummies = pd.get_dummies(total_frame['mechanics'], prefix='mech', drop_first=True)
total_frame = pd.concat([total_frame, mech_dummies], axis=1)
type_dummies = pd.get_dummies(total_frame['type'], prefix='type', drop_first=True)
total_frame = pd.concat([total_frame, type_dummies], axis=1)
```

In [10]:
```
# preserve 'total frame' for testing with all years later
tree_frame = total_frame[total_frame['year'] >= break_year]
```

filter out categorical columns for tree fitting

In [11]:
```
# list of just desired features, now including one-hot cols and remove categorical cols
features = list(tree_frame.columns)

# remove categorical cols
features.remove('mechanics')
features.remove('type')
```

Divide the data set

split data into training portions

In [12]:
```
# filtered features with one-hot fixes for categorical columns
X = tree_frame[features]

# target variable
y = tree_frame[['categories']]

# Split method, 0.3 == 30% of data saved for testing, choosen randomly from set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4
```

split training data so model can "learn" different categories separately

keep testing data intact for checking accuracy

In [13]:
```
# split training data by category
X_train = split_data_frame_list(X_train, 'categories')
y_train = split_data_frame_list(y_train, 'categories')

# remove target column from X_data
# typically done before, but was needed for splitting training data
X_train.drop('categories', inplace=True, axis=1)
X_test.drop('categories', inplace=True, axis=1)
```

Train the model

adjust variables prior to loop for hyperparameter adjustments

In [14]:
```
from copy import deepcopy

best_accuracy = 0
models_to_compare = 3
tree_depth = 20
for j in range(models_to_compare):
    # Decision Tree classifer object
    dtc = DecisionTreeClassifier(criterion="entropy", splitter='best', max_depth=tree_d

    # Train Decision Tree Classifer
    dtc = dtc.fit(X_train,y_train)

    # predictions by model for y
    y_pred = dtc.predict(X_test)

    # custom accuracy check
    correct = 0
    y_targets = y_test["categories"].tolist()
    for i in range(len(y_pred)):
        if y_pred[i] in y_targets[i]:
            correct += 1

    accuracy = correct / len(y_pred)
    if accuracy > best_accuracy:
        best_dtc = deepcopy(dtc)
        best_accuracy = accuracy
        print("best accuracy: ", accuracy)

print('done!')
```

```
best accuracy:  0.8910792883268241
best accuracy:  0.891513235385283
done!
```

Save model created in loop above

In [18]:
```
from joblib import dump, load

# swtich to model directory
model_dir = working_dir + "\models"
os.chdir(model_dir)

# create and save file
# [model type]_[deatils]_[accuracy]
joblib_file = "dtc_test_8752.joblib"
dump(best_dtc, joblib_file)
```

Out[18]: ['dtc_test_8752.joblib']

load and test model

In [19]:
```
# swtich to model directory and load
model_dir = working_dir + "\models"
os.chdir(model_dir)

# name of model in folder
joblib_file = "dtc_test_8752.joblib"
joblib_model = load(joblib_file)
```

get new set not limited by year (if desired for 'complete' check)

In [20]:
```
# before year trimmed
X = total_frame[features]
y = total_frame[['categories']]
```

split data into new set to compare against

In [21]:
```
# percent of data to use for test
pod = 50

# split into new set to test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=(pod/100), random_s

# then remove category column
X_test.drop('categories', inplace=True, axis=1)
```

test loaded model's accuracy

In [22]:
```
y_pred = joblib_model.predict(X_test)

correct = 0
y_targets = y_test["categories"].tolist()
for i in range(len(y_pred)):
    if y_pred[i] in y_targets[i]:
        correct += 1

print("Accuracy:", correct / len(y_pred))
```

```
Accuracy: 0.9615815233037971
```

Score Prediction Regression Model

In [1]:
```
import os
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures

working_dir = os.getcwd()
game_data = pd.read_csv('game_data.csv')
game_data.head() # See the first 5 rows to check data import
```

Out[1]:

	id	type	name	year	minplayers	maxplayers	playingtime	minplaytime	maxplayti
0	13	boardgame	Catan	1995	3	4	120	60	1
1	822	boardgame	Carcassonne	2000	2	5	45	30	
2	30549	boardgame	Pandemic	2008	2	4	45	45	
3	68448	boardgame	7 Wonders	2010	2	7	30	30	
4	36218	boardgame	Dominion	2008	2	4	30	30	

5 rows × 22 columns

clean and filter data

selective filtering for removing extremes, retain 'game_data' in case any reference to complete set is needed

In [2]:
```
bgg_games = game_data[game_data['year'] > 1980]
bgg_games = bgg_games[bgg_games['maxplayers'] <= 30]
bgg_games = bgg_games[bgg_games['minplaytime'] <= 180] # 120 - 90th percentile
bgg_games = bgg_games[bgg_games['maxplaytime'] <= 720]
bgg_games = bgg_games[bgg_games['minage'] <= 21]
```

```
bgg_games = bgg_games[bgg_games['playingtime'] >= 10]
bgg_games = bgg_games[bgg_games['maxplayers'] >= bgg_games['minplayers']]
```

select cells potentially relevant to rating (before community interaction)

In [3]:
```
dtc_test = bgg_games[['type', 'minplayers', 'maxplayers', 'playingtime',
       'minplaytime', 'maxplaytime', 'minage', 'avg_rating', 'mechanics',
       'bay_rating', 'total_comments', 'total_weights', 'complexity', 'categories']]
```

convert mechanics and categories into lists with values

In [4]:
```
dtc_test['categories'] = dtc_test['categories'].apply(lambda x: x.strip(']['').split(',
dtc_test['mechanics'] = dtc_test['mechanics'].apply(lambda x: x.strip(']['').split(', ')
```

count number of mechanics and categories for each game, make new columns

In [5]:
```
dtc_test['num_mechs'] = dtc_test.apply(lambda row: len(row['mechanics']), axis=1)
dtc_test['num_cats'] = dtc_test.apply(lambda row: len(row['categories']), axis=1)
dtc_test['player_diff'] = dtc_test.maxplayers - dtc_test.minplayers
```

In [6]:
```
def split_data_frame_list(df, target_column, output_type=str):
    '''
    Accepts a column with list values and splits into several rows.

    df: dataframe to split
    target_column: the column containing the values to split
    output_type: type of all outputs
    returns: a dataframe with each entry for the target column separated, with each ele
    The values in the other columns are duplicated across the newly divided rows.
    '''
    row_accumulator = []

    def split_list_to_rows(row):
        split_row = row[target_column]
        if isinstance(split_row, list):
          for s in split_row:
              new_row = row.to_dict()
              new_row[target_column] = output_type(s)
              row_accumulator.append(new_row)
        else:
          new_row = row.to_dict()
          new_row[target_column] = output_type(split_row)
          row_accumulator.append(new_row)

    df.apply(split_list_to_rows, axis=1)
    new_df = pd.DataFrame(row_accumulator)

    return new_df
```

split lists into multiple rows for regression model

In [7]:
```
dtc_test = split_data_frame_list(dtc_test, 'categories')
dtc_test = split_data_frame_list(dtc_test, 'mechanics')
```

get all desired cols and apply one-hot fix to categorical features

In [8]:
```
# for filtering cols
desired_cols = ['type', 'minplayers', 'maxplayers', 'playingtime', 'minplaytime', 'maxp
```

```
# make dummies (one-hot fix) for categorical values
# will remove categorical columns as well
model_frame = dtc_test[desired_cols]
model_frame = pd.get_dummies(model_frame, drop_first=True)
```

filter out columns for model fitting

In [9]:
```
# list of just desired features, now including one-hot cols
features = list(model_frame.columns)

# remove target cols
features.remove('avg_rating')
features.remove('bay_rating')
```

Divide the data set

split data into training and testing portions

In [10]:
```
# filtered features with one-hot fixes for categorical columns
X = model_frame[features]

# remove undesired columns
X.drop("categories_'Expansion for Base-game'", inplace=True, axis=1) # duplicate of "is

# target variable - average because bays is weighted so heavily
y = model_frame[['avg_rating']]

# Split data - limit training because it will be filtered and re-allocated later
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.0001, random_stat
```

feature selection

In [11]:
```
from sklearn.feature_selection import SelectKBest, f_regression, mutual_info_regression
from functools import partial

# f_regression is univarite - direct correlations
# mutual_info compares multiple feature pairs

def feature_selection(X_train, y_train, m=0, n='all'):
        '''produces feature values to help with selection
        IN: 3 frames split from data
                m = type of method desired (int)
                n = number of top features to select
        OUT: transformed X-data and feature selection model'''

        # partial to establish params for mutual info
        # CAN'T HANDLE
        mutual_info = partial(mutual_info_regression, random_state=0)

        # scoring functions to use
        methods = [f_regression, mutual_info]

        # configure to select type of feature value grader and number of top features t
        fs = SelectKBest(score_func=methods[m], k=n)

        # correlate relationships from training data
        fs.fit(X_train, y_train)
```

```
        return fs
```

feature selection parameters for filtering columns into regression model

In [12]:
```
# feature selection scores (currently set to check all features)
fs = feature_selection(X_train, y_train, 0, 15)
feature_mask = fs.get_support()
top_features = X.columns[feature_mask]
```

reset test data for mutual information filtering

In [13]:
```
X = model_frame[top_features]

# still the same
y = model_frame[['avg_rating']]

# new data split for training with limited set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=2
```

In [14]:
```
# # DANGER DANGER
# # new feature selection model
# fs_mut = feature_selection(X_train, y_train, 1, 15)
# feature_mask = fs.get_support()
# top_features = X.columns[feature_mask]
# print(top_features)
```

Train the model

In [17]:
```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from copy import deepcopy
import math

# for checking accuracy
best_r2 = 0
best_mse = math.inf

# random forest regressor creates many regression trees for evaluation
rfr = RandomForestRegressor(n_estimators=100, max_depth=20, random_state=42)

# fit model to training data
rfr.fit(X_train,y_train)

# predictions by model for y
y_pred = rfr.predict(X_test)

# accuracy check, lower is better
mse = mean_squared_error(y_test, y_pred)
print('MSE Forest: ', mse)

# The coefficient of determination: 1 is perfect prediction
r2 = r2_score(y_test, y_pred)
print('r2 Forest: ', r2)
```

```
MSE Forest:  0.050877176437359596
r2 Forest:  0.9411821686719545
```

In [18]:
```
# check specific instances for accuracy
target = 170 # must be smaller than dataframe length and >=0
print("target score: ", y_test.iloc[target])
print("predicted score: ", y_pred[target])
```

```
target score:  avg_rating    8.22286
Name: 42866, dtype: float64
predicted score:  8.218860365220493
```

save model and test accuracy of accuracy rating

In [19]:
```
from joblib import dump, load

# swtich to model directory
model_dir = working_dir + "\models"
os.chdir(model_dir)

# create and save file
# [model type]_[details]_[accuracy]
joblib_file = "rfr_test_9412.joblib"
dump(rfr, joblib_file)

# test load
joblib_model = load(joblib_file)

y_pred = joblib_model.predict(X_test)

# accuracy check
mse = mean_squared_error(y_test, y_pred)
print('MSE: ', mse)

r2 = r2_score(y_test, y_pred)
print('r2: ', r2)
```

```
MSE:  0.050877176437359596
r2:  0.9411821686719545
```

load and test model

In [20]:
```
# swtich to model directory
model_dir = working_dir + "\models"
os.chdir(model_dir)

# file name with model to load
joblib_file = "rfr_test_9412.joblib"
joblib_model = load(joblib_file)
```

re-load data to check against different set, after initial test

(needs to be run twice for some reason...)

In [23]:
```
X = model_frame[top_features]

# still the same
y = model_frame[['avg_rating']]

# new data split for training with limited set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=2
```

re-test model

In [24]:
```
# model prediction scores
y_pred = joblib_model.predict(X_test)

# accuracy check
mse = mean_squared_error(y_test, y_pred)
print('MSE: ', mse)

r2 = r2_score(y_test, y_pred)
print('r2: ', r2)
```

```
MSE:  0.039372519846675114
r2:  0.9547505971722409
```