```
# -*- coding: utf-8 -*-
"""DT_Class_Lab.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/183up4skWA0yG8tgql1lzsaRTM5N0yz35

Congrats! You just graduated UVA's BSDS program and got a job working at a movie studio in Hollywood.

Your boss is the head of the studio and wants to know if they can gain a competitive advantage by predicting new movies that might get high imdb scores (movie rating).

You would like to be able to explain the model to mere mortals but need a fairly robust and flexible approach so you've chosen to use decision trees to get started.

In doing so, similar to  great data scientists of the past you remembered the excellent education provided to you at UVA in a undergrad data science course and have outline 20ish steps that will need to be undertaken to complete this task. As always, you will need to make sure to #comment your work heavily.

 Footnotes:
-     You can add or combine steps if needed
-     Also, remember to try several methods during evaluation and always be mindful of how the model will be used in practice.
- Make sure all your variables are the correct type (factor, character,numeric, etc.)
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import graphviz
from sklearn.model_selection import
train_test_split,GridSearchCV,RepeatedStratifiedKFold
from sklearn import metrics
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import plot_tree, DecisionTreeClassifier, export_graphviz

#1. Load the data
#Sometimes need to set the working directory back out of a folder that we create a file in
from google.colab import drive
drive.mount('/content/drive')
import pandas as pd
file_path = '/content/drive/My Drive/movie_metadata.csv'

"""2 Ensure all the variables are classified correctly including the target variable and collapse factor variables as needed.

3 Check for missing variables and correct as needed. Once you've completed the cleaning again create a function that will do this for you in the future. In the submission, include only the function and the function call.
"""

def preprocess_moviedata(df):
```

```
  df = df.dropna() # can drop all na vals because dataset large

  drop = [
        'num_critic_for_reviews', 'actor_2_name', 'genres', 'actor_1_name',
        'movie_title', 'num_voted_users', 'actor_3_name', 'plot_keywords',
        'movie_imdb_link', 'num_user_for_reviews', 'director_name',
        'actor_1_facebook_likes', 'actor_2_facebook_likes',
        'actor_3_facebook_likes', 'aspect_ratio'
    ]
    # columns to drop (either irrelevant, explained by a different column, or
unexplained by lack of data dictionary)

  df = df.drop(columns = drop)

  # fix content_rating

  df['content_rating'] = df['content_rating'].replace({
        "Not Rated": "Unrated",
        "X": "NC-17" }) # X is outdated version of NC-17 (thanks google)


  # collapse other outdated categories with no straightforward map
  df['content_rating'] = df['content_rating'].apply(
        lambda x: x if x in ["R", "PG-13", "PG", "NC-17", "Unrated"] else "Other" )

  # label encode categorical vars (do not need to convert to categorical because
sklearn's label encoder can handle string/object types)
  cat_cols = df.select_dtypes(include="object").columns


  for i in cat_cols:
      df[i] = LabelEncoder().fit_transform(df[i])

  # last handle target variable (imdb_rating)
  mean_rating = df['imdb_score'].mean()
  df['imdb_score'] = df['imdb_score'].apply(lambda x: 'good' if x > mean_rating
else 'bad')

  df['imdb_score'] = df['imdb_score'].replace(
      {
          'good' : 1,
          'bad' : 0
      }
  )

  return(df)

raw_movie_data = pd.read_csv(file_path)
movie_data = preprocess_moviedata(raw_movie_data)
print(movie_data.info())

"""4 Guess what, you don't need to scale the data, because DTs don't require this
to be done, they make local greedy decisions...keeps getting easier, go to the next
step.

5 Determine the baserate or prevalence for the classifier, what does this number
mean?
"""
```

```
# print(movie_data['imdb_score'].value_counts())

# prevalence = # positive / total #
prevalence = sum(movie_data['imdb_score']==1)/ len(movie_data['imdb_score'])
prevalence

# The prevalence being 0.547 means that the prevalence of the positive class (in
this case, the probability movie being labelled as 'good') is 54.7%
# This means that if you guess at random you'd be right 54.7% of the time
# so the decision tree should aim to beat that

"""6 Split your data into test, tune, and train. (80/10/10)"""

from sklearn.model_selection import train_test_split
X = movie_data.drop(columns = ['imdb_score'])
y = movie_data['imdb_score']

# first find trains and temps to use in finding tune/test
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y, train_size=0.80, stratify=y, random_state=34) # choose any random val for
random_state for reproducibility

X_tune, X_test, y_tune, y_test = train_test_split(
    X_temp, y_temp, train_size=0.50, stratify=y_temp, random_state=92 # change
random seed so its not the same as above
) # use train_size = 0.5 because its half of the remaining 20% (so 10/10)

# now you have X_train, y_train, X_tune, y_tune, X_test, y_test

"""7 Create the kfold object for cross validation."""

kf = RepeatedStratifiedKFold(n_splits=10, n_repeats=5, random_state=32) # from the
wine quality example but applicable to any generally large dataset like movie_data

"""8 Create the scoring metric you will use to evaluate your model and the max
depth hyperparameter (grid search)"""

scoring = ['precision', 'f1', 'balanced_accuracy'] # created scoring metric like in
the notes but selected different metrics than the example

"""9 Build the classifier object"""

classifier= DecisionTreeClassifier(random_state=352)

"""10 Use the kfold object and the scoring metric to find the best hyperparameter
value for max depth via the grid search method."""

params = {'max_depth': range(1, 11)} # will try max_depth values of 1 through 10
for the tree to find the best #

gridsearch = GridSearchCV(classifier, params, scoring=scoring, cv=kf,refit='f1')

"""11 Fit the model to the training data."""

model = gridsearch.fit(X_train, y_train)

"""12 What is the best depth value?"""
```

```
best_depth = model.best_estimator_
best_depth

"""13 Print out the model"""

plt.figure(figsize=(20,10))
plot_tree( # plot decision tree
    best_depth,
    feature_names=X_train.columns,
    class_names=['bad', 'good'],  # <- adjust if you have different class names
    filled=True,
    rounded=True
)
plt.show()

"""14 View the results, comment on how the model performed using the metrics you
selected."""

# find the 3 metrics used in scoring from above
f1 = model.cv_results_['mean_test_f1']
precision = model.cv_results_['mean_test_precision']
bal_acc = model.cv_results_['mean_test_balanced_accuracy']

# find sd for comparison like in notes
SDf1 = model.cv_results_['std_test_f1']
SDprecision = model.cv_results_['std_test_precision']
SDbal_acc = model.cv_results_['std_test_balanced_accuracy']

depth = list(range(1, 11)) # use all values of max_depth tested, which was just 1
through 10

# create data frame to show model and its results
results_model = pd.DataFrame(
    list(zip(depth, f1, precision, bal_acc, SDf1, SDprecision, SDbal_acc)),
    columns=['depth', 'f1', 'precision', 'balanced_accuracy', 'f1SD',
'precisionSD', 'balanced_accuracySD']
)
print(results_model.head(10)) # prints results for all 10 depths

# as you can see from the printed df, depth = 7 has the highest f1 score (meaning
model balances precision and recall best at max_depth=7)
# The model's precision did best at depth = 6 but depth = 7 was not far behind
# The best balanced accuracy (which is the mean of sensitivity and specificity)
also came from depth = 7
# The sds are all low meaning the model is reliable across the 10 folds

"""15 Which variables appear to be contributing the most (variable importance)"""

varimp = pd.DataFrame(
    best_depth.feature_importances_, # finds importances for the optimized model
    index=X_train.columns,
    columns=['importance']
).sort_values('importance', ascending=False) # show them with max importance at top

print(varimp)
# this shows that duration and movie's facebook likes contributes the most to the
rating, followed by budget, gross, and title_year
# Intuitively this makes a lot of sense. color has low importance likely due to the
fact that the majority of the data was in color and not black and white
```

```
"""16 Use the predict method on the tune data and print out the results."""

tune_pred = best_depth.predict(X_tune)
print(tune_pred)

"""17 How does the model perform on the tune data?"""

# i will evaluate using the same metrics as before
from sklearn.metrics import f1_score, precision_score, balanced_accuracy_score

f1 = f1_score(y_tune, tune_pred)
precision = precision_score(y_tune, tune_pred)
bal_acc = balanced_accuracy_score(y_tune, tune_pred)

print (f1, precision, bal_acc)
# these are pretty good scores (all above 0.7) so the model performs well on the
tune data

"""18 Print out the confusion matrix for the tune data, what does it tell you about
the model?"""

print(ConfusionMatrixDisplay.from_estimator(best_depth,X_tune,y_tune,
display_labels = ['bad','good'], colorbar=False))

"""This shows that the model does best at performing true positives (143). The most
prevalent mistake is false negatives (62), meaning the model is sooner to predict
'bad' for a movie than 'good'. If the goal is to find/focus on higher quality
movies, this is a good stricter model.

19 What are the top 3 movies based on the tune set? Which variables are most
important in predicting the top 3 movies?
"""

# get probabilities for positive (good movies) class
tune_probs = best_depth.predict_proba(X_tune)[:, 1]

# im going to make a new df to add columns for the predicted values and their
probabilities of being correct
newDf = X_tune.copy()
newDf["predicted_label"] = tune_pred
newDf["prob_good"] = tune_probs

# add back in title column since deleted it earlier
newDf["movie_title"] = raw_movie_data.loc[newDf.index, "movie_title"]

# select top 3 by highest probability/confidence
top_3 = newDf[newDf["predicted_label"] == 1].sort_values("prob_good",
ascending=False).head(3)

print(top_3['movie_title'])
# this is interesting because Star Wars movie is probably highly rated so it did
well

# The same variables are the most important in predicting the top 3 movies as in
Q15.
# Specifically, the top three most contributing variables are duration,
movie_facebook_likes, and budget.
```

```
"""20 Use a different hyperparameter for the grid search function and go through
the process above again using the tune set."""

# referenced powerpoint notes to select new hyperparameter: Minimum samples for a
terminal node (leaf)
new_param_grid = {'min_samples_leaf': range(1, 11)}  # trying leaf sizes from 1 to
10

new_gridsearch = GridSearchCV(
    estimator=DecisionTreeClassifier(random_state=754),
    param_grid=new_param_grid,
    scoring=scoring, # same
    cv=kf, # same as before
    refit='f1')     # same

# train on train set
new_gridsearch.fit(X_train, y_train)
best_samples = new_gridsearch.best_estimator_ # optimized parameter for minimum
samples for leaf

# now use on tune set
tune_new = best_samples.predict(X_tune)

# find metrics

f1_new = f1_score(y_tune, tune_new)
precision_new = precision_score(y_tune, tune_new)
bal_acc_new = balanced_accuracy_score(y_tune, tune_new)

print(f1_new)
print(precision_new)
print(bal_acc_new)

"""21 Did the model improve with the new hyperparameter search?

Yes. <br>
Original f1, precision, and balance accuracy values (rounded):
0.72, 0.74, 0.70 <br>
New f1, precision, and balance accuracy values (rounded):
0.74, 0.75, 0.72 <br>

Higher values are better for all 3 metrics, so the increase with the new model
shows that the model performs better with the new hyperparameter search.

(Note: I looked up how to make a new line in colab because it was messy looking)

22 Using the better model, predict the test data and print out the results.
"""

best_model = best_samples

preds = best_model.predict(X_test)

print(preds)

"""23 Summarize what you learned along the way and make recommendations to your
boss on how this could be used moving forward, being careful not to over promise.

Along the way, I first learned which features might hold significance to the model,
```

and dropped irrelevant features. I preprocessed the data through label encoding, and then built my first model with hyperparameter maximum depth. This worked fairly well to predict whether a movie was 'good' or 'bad' (based on a mean threshold I established in the preprocessing stage), but through exploring a second hyperparameter, I was able to improve the results.

Through this exploration, I learned the importance of adjusting parameters. Through changing the hyperparameter from max depth to minimum # of samples for a leaf, I was able to create a model that performed more accurately. Ideally, it would be helpful to test all hyperparameters (including minimum samples for a node split, maxmimum number of terminal nodes, and maximum features to consider for split) to optimize a model's performance.

Recommendations:

The most straightforward way to potentially predict (or produce) a movie with a higher rating is to optimize movie length. Since movie length (duration) is the most influential variable in my model, the data could be explored to vie the range of durations that produce the most popular movies (this is a simple calculation). The correlation between facebook likes and higher ratings may indicate that a strong social media campaign/internet presence in promoting the movie may contribute to its success. Budget also contributes highly to the model, though this is intuitive, and less easy to control.
"""