

# Stellar Classification

## Imports

In [1]:

```
import numpy as np
import pandas as pd
from matplotlib import rcParams
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import zscore
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, export_graphviz
import graphviz
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler, OneHotEncoder, minmax_scale
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_re
```

## Read the data and take a first look

In [2]:

```
df = pd.read_csv('https://raw.githubusercontent.com/Deelane/CST383---Final/main/star_cl
```

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 18 columns):
 #   Column      Non-Null Count  Dtype  
 --- 
 0   obj_ID      100000 non-null   float64
 1   alpha        100000 non-null   float64
 2   delta        100000 non-null   float64
 3   u            100000 non-null   float64
 4   g            100000 non-null   float64
 5   r            100000 non-null   float64
 6   i            100000 non-null   float64
 7   z            100000 non-null   float64
 8   run_ID      100000 non-null   int64  
 9   rerun_ID    100000 non-null   int64  
 10  cam_col     100000 non-null   int64  
 11  field_ID    100000 non-null   int64  
 12  spec_obj_ID 100000 non-null   float64
 13  class        100000 non-null   object 
 14  redshift     100000 non-null   float64
 15  plate        100000 non-null   int64  
 16  MJD          100000 non-null   int64  
 17  fiber_ID    100000 non-null   int64  
dtypes: float64(10), int64(7), object(1)
memory usage: 13.7+ MB
```

The dataset has no null entries, but we will still need to check for outliers

In [4]:

```
df.describe()
```

Out[4]:

	<b>obj_ID</b>	<b>alpha</b>	<b>delta</b>	<b>u</b>	<b>g</b>	<b>r</b>	<b>i</b>
<b>count</b>	1.000000e+05	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000
<b>mean</b>	1.237665e+18	177.629117	24.135305	21.980468	20.531387	19.645762	18.135828
<b>std</b>	8.438560e+12	96.502241	19.644665	31.769291	31.750292	1.854760	1.854760
<b>min</b>	1.237646e+18	0.005528	-18.785328	-9999.000000	-9999.000000	9.822070	9.822070
<b>25%</b>	1.237659e+18	127.518222	5.146771	20.352353	18.965230	18.135828	18.135828
<b>50%</b>	1.237663e+18	180.900700	23.645922	22.179135	21.099835	20.125290	20.125290
<b>75%</b>	1.237668e+18	233.895005	39.901550	23.687440	22.123767	21.044785	21.044785
<b>max</b>	1.237681e+18	359.999810	83.000519	32.781390	31.602240	29.571860	29.571860

Note the -9999 values as min for columns u, g, and z. Is this missing/bad data?

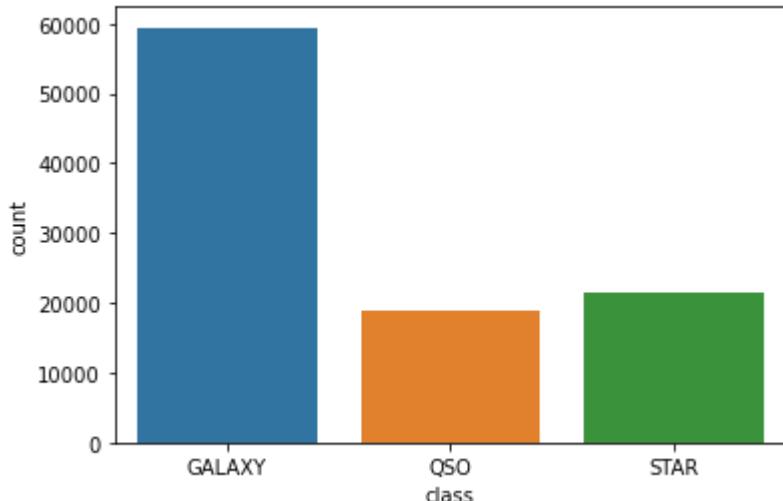
In [5]:

```
print(df['class'].value_counts())
sns.countplot('class', data=df);
bad_index = df[df['g'] == -9999].index[0]
df.drop([bad_index], inplace=True) #drop the bad data
df.reset_index(drop=True, inplace=True)
```

```
GALAXY      59445
STAR        21594
QSO         18961
Name: class, dtype: int64
```

C:\Users\Jordan\anaconda3\lib\site-packages\seaborn\\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

```
warnings.warn(
```

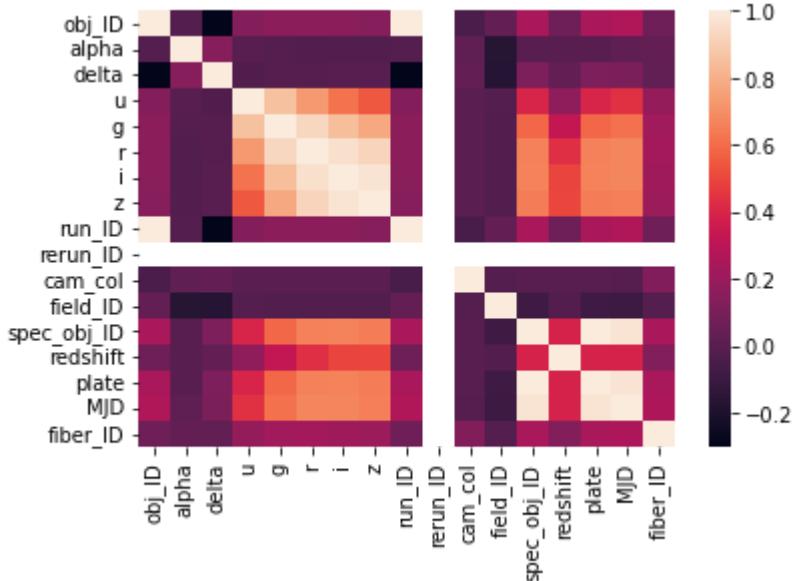


As you can see, this is an unbalanced dataset. There are significantly more 'GALAXY' entries than

'QSO' and 'STAR'

In [6]:

```
corr = df.corr()
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns);
```



## Data Visualization / Preprocessing:

### Dropping ID columns

In [7]:

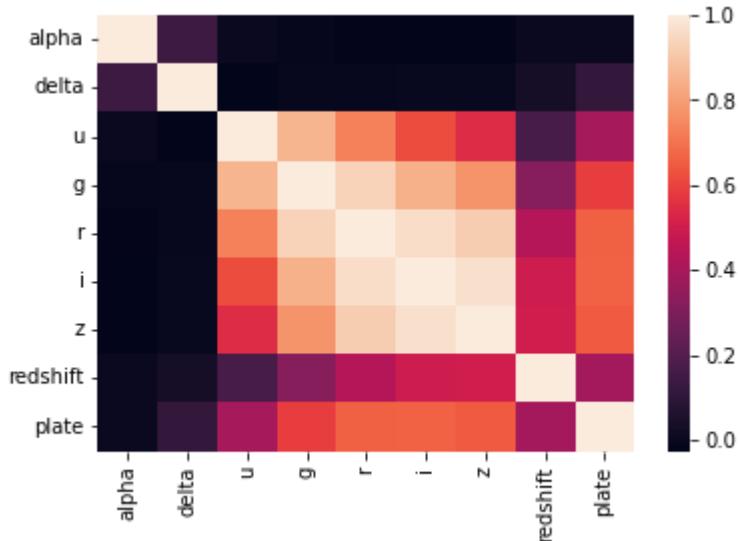
```
df.drop(columns=['obj_ID', 'run_ID', 'rerun_ID', 'field_ID', 'fiber_ID', 'spec_obj_ID',
```

### Remaining columns

Columns ['u', 'g', 'z'] and ['r', 'i'] correlate to the other columns in almost exactly same way. 'r' and 'i' are different enough with regards to redshift (our biggest predictor) so we will keep them.

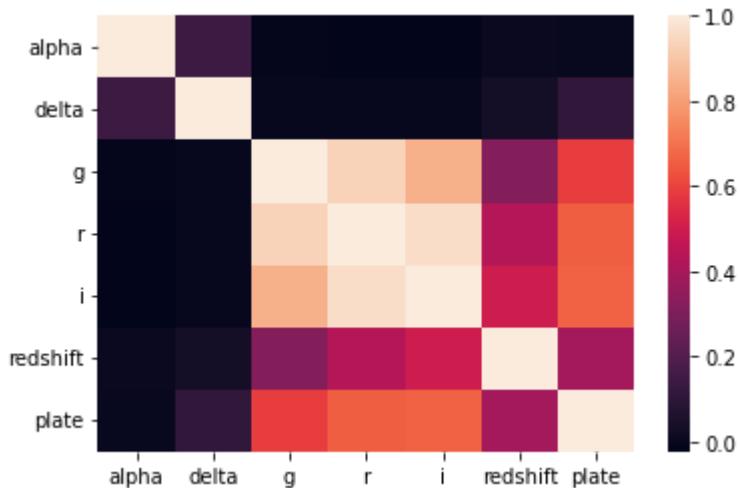
In [8]:

```
sns.heatmap(df.corr());
```



## Final columns

```
In [9]: df.drop(columns=['u', 'z'], inplace=True)
sns.heatmap(df.corr());
```

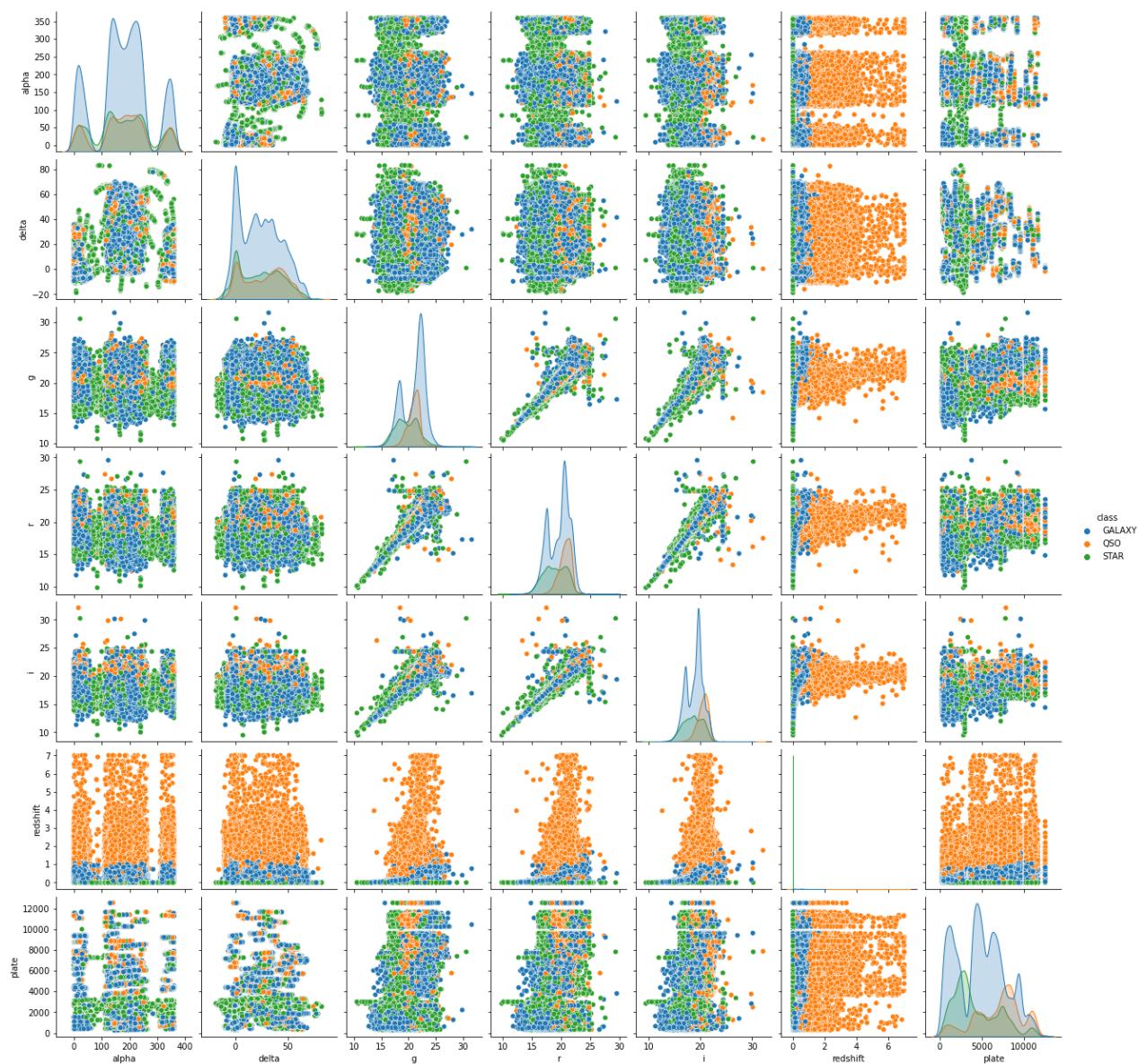


## Pairplot

'redshift', 'g', and 'redshift' vs 'g' show to be very good at classifying, with each class clearly having its own value range for each.

```
In [10]: sns.pairplot(df, hue='class')
```

```
Out[10]: <seaborn.axisgrid.PairGrid at 0x211eb05a6a0>
```



## One Hot Encoding

In [11]:

```
enc = OneHotEncoder()
enc_df = pd.DataFrame(enc.fit_transform(df[['class']]).toarray())
enc_df.columns = ['GALAXY', 'QSO', 'STAR']
df = pd.concat([df, enc_df], axis=1)
```

## Forward Selection - Manual Testing

### Selecting Predictors

In [12]:

```
targets = ['GALAXY', 'QSO', 'STAR']

# run 1
# predictors = ['alpha', 'delta', 'g', 'r', 'i', 'redshift', 'plate']

# run 2
# redshift provides highest accuracy:
# predictors = ['alpha', 'delta', 'g', 'r', 'i', 'plate']
```

```
# run 3
# g + redshift provides highest accuracy:
# predictors = ['alpha', 'delta', 'r', 'i', 'plate']

# run 4
# r + g + redshift provides highest accuracy:
# predictors = ['alpha', 'delta', 'i', 'plate']

# run 5
# i + r + g + redshift provides highest accuracy:
predictors = ['alpha', 'delta', 'plate']
```

## Getting Accuracy

In [13]:

```
for p in predictors:
    # run 1 - for all predictors:
    # X = df[[p]].values

    # run 2
    # redshift provides highest accuracy: 94.31, add to predictors
    # X = df[[p, 'redshift']].values

    # run 3
    # g + redshift provides highest accuracy: 95.4, add to predictors
    # X = df[[p, 'redshift', 'g']].values

    # run 4
    # r + g + redshift provides highest accuracy: 95.08, add to predictors
    # X = df[[p, 'redshift', 'g', 'r']].values

    # run 5
    # i + r + g + redshift provides highest accuracy: 93.86
    X = df[[p, 'redshift', 'g', 'r', 'i']].values

y = df[targets].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# implement kNN classification with default value of k
knn = KNeighborsClassifier(weights='distance', metric='manhattan')
knn.fit(X_train, y_train)

# get predictions
predictions = knn.predict(X_test)

predictions_enc = pd.DataFrame(predictions).idxmax(axis=1)
y_test_enc = pd.DataFrame(y_test).idxmax(axis=1)

print (p, (predictions_enc == y_test_enc).mean())
```

```
alpha 0.9565
delta 0.9546
plate 0.9620666666666666
```

## Set Predictors and Targets

In [14]:

```
# Set predictors and targets
predictors = ['alpha', 'delta', 'g', 'r', 'i', 'redshift', 'plate']
targets = ['GALAXY', 'QSO', 'STAR']

X = df[predictors].values
y = df[targets].values
```

## Train, Test, Split

In [15]:

```
#Train, Test, Split (Random Seed 42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4)
```

## Choosing a model

### KNN Exploration

In [16]:

```
#WARNING, COSTLY COMPUTATIONS

#use gridsearch on cross validation to find the best n and p for knn
n_sqrt = round(np.sqrt(X_train.shape[0]))
grid = [{n_neighbors: [3, 5, 7, 9, n_sqrt], 'p':[1,2]}]
knnCV = GridSearchCV(KNeighborsClassifier(), grid, cv=5, scoring='accuracy')
knnCV.fit(X_train, y_train)
print(knnCV.best_params_)

#Best hyperparameters evaluate to {'n_neighbors': 3, 'p': 1}, so 3 neighbors and the di
#This will likely change as it is possible that we are overfitting

{'n_neighbors': 3, 'p': 1}
```

### Learning Curves

In [17]:

```
from sklearn.model_selection import learning_curve

#cell height
# from IPython.display import Javascript
# display(Javascript('''google.colab.output.setIframeHeight(0, true, {maxHeight: 5000})'''))

#WARNING, EXTREMELY COSTLY COMPUTATIONS
#14mins on local, likely much more on colab

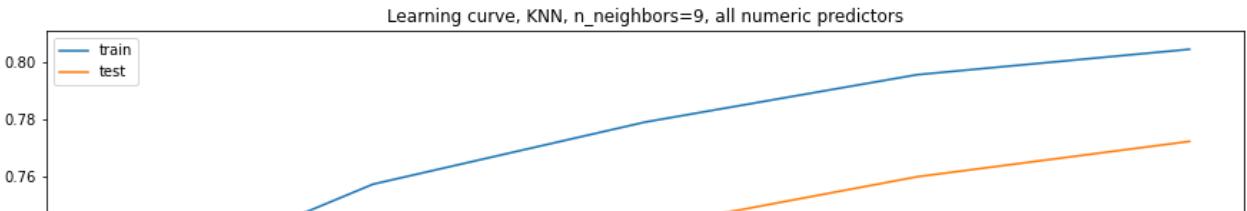
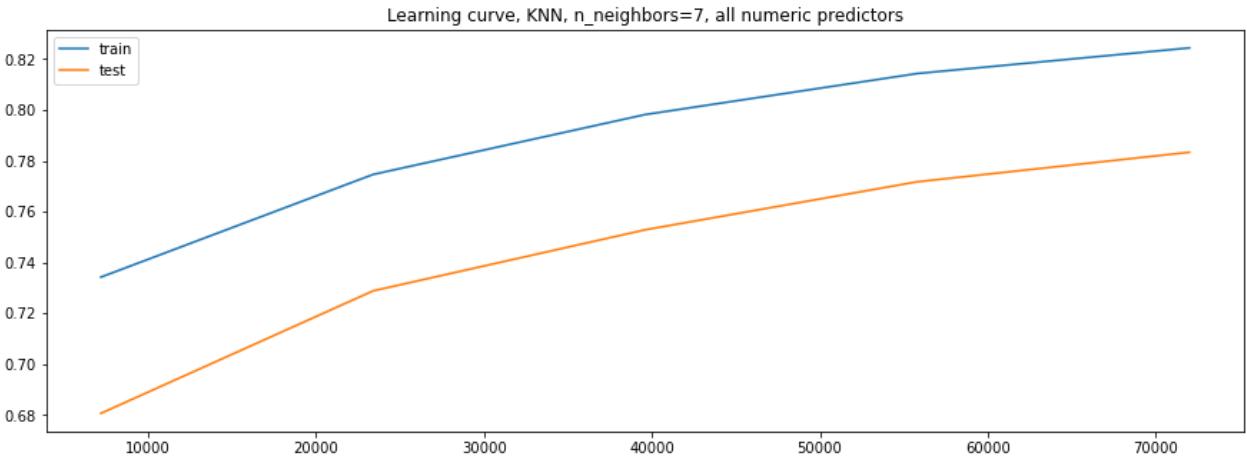
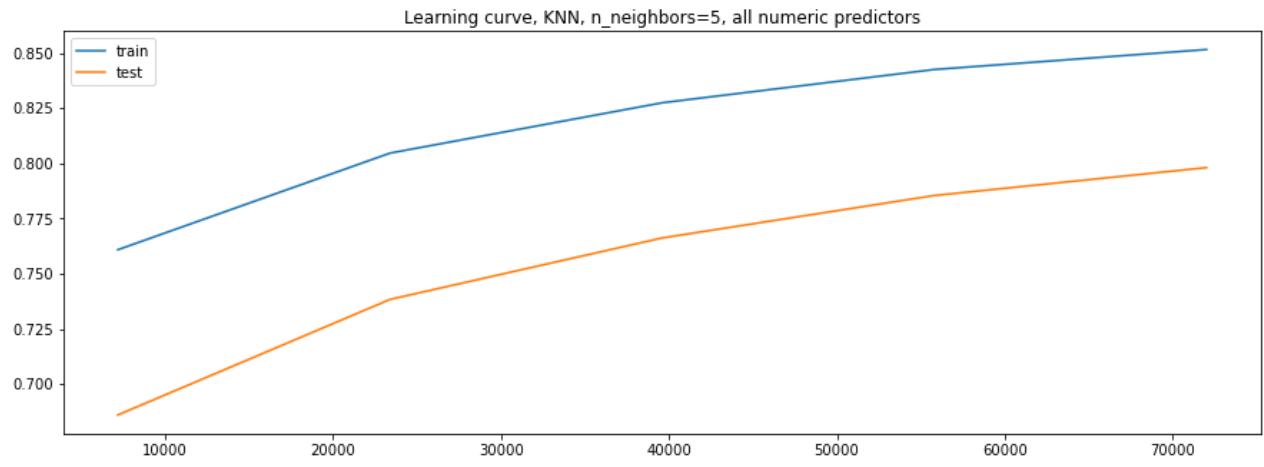
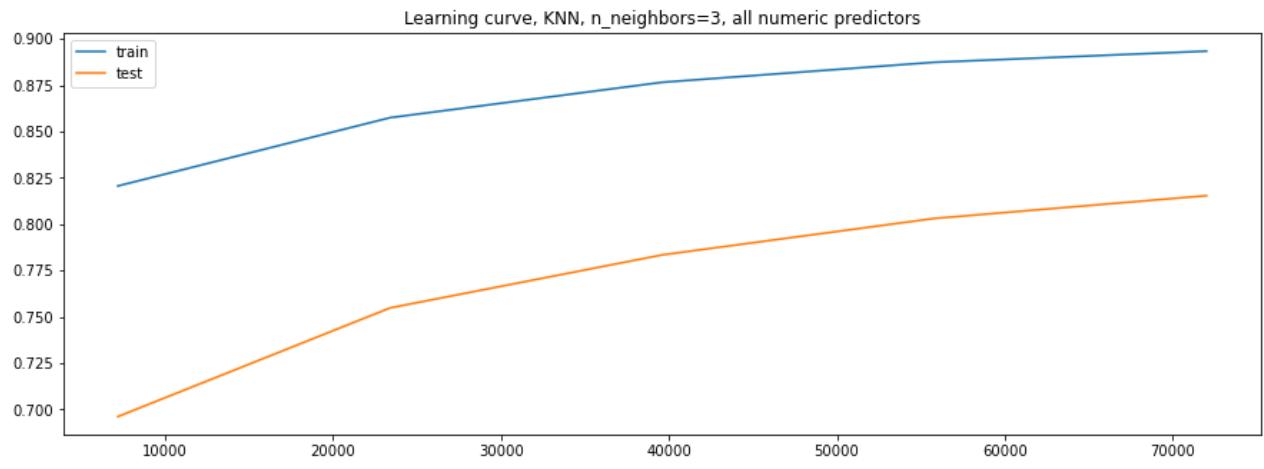
n_neighbors_list = [3, 5, 7, 9, round(np.sqrt(X_train.shape[0]))]
fig, axs = plt.subplots(len(n_neighbors_list), figsize=(15,30))
learning_curve_df = pd.DataFrame({ 'n_neighbors':[], 'train_scores_mean':[], 'test_scor

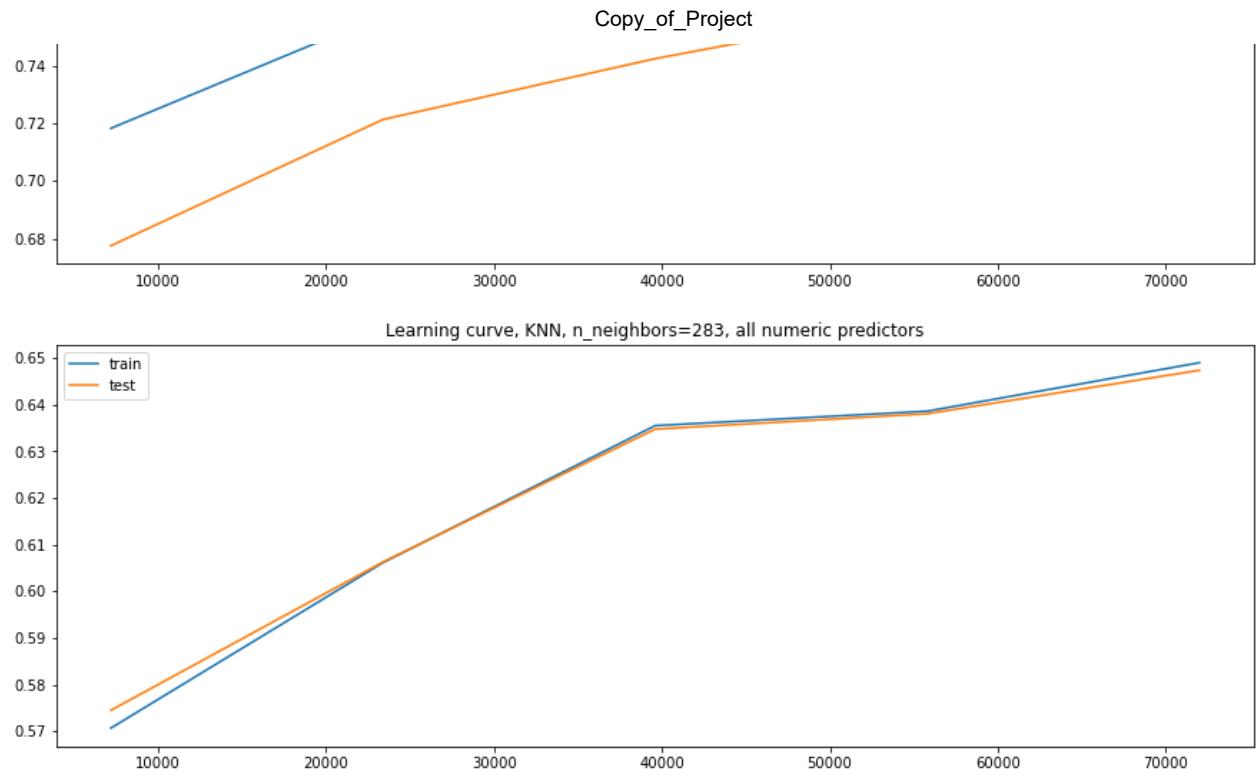
for i in range(len(n_neighbors_list)):
    knn = KNeighborsClassifier(n_neighbors = n_neighbors_list[i], p = 1) #Manhattan dis
    train_sizes, train_scores, test_scores = learning_curve(knn, X_train, y_train, cv=1
    train_scores_mean = np.mean(train_scores, axis=1)
```

```

test_scores_mean = np.mean(test_scores, axis=1)
df_temp = pd.DataFrame([[n_neighbors_list[i], train_scores_mean, test_scores_mean]])
learning_curve_df = pd.concat([learning_curve_df, df_temp], ignore_index=True)
axs[i].plot(train_sizes, train_scores_mean, label='train')
axs[i].plot(train_sizes, test_scores_mean, label='test')
axs[i].legend()
title = "Learning curve, KNN, n_neighbors=" + str(n_neighbors_list[i]) + ", all numeric predictors"
axs[i].set_title(title)

```





## Logistic Regression Exploration

### Label Encoding

```
In [18]: df['class_cat'] = df['class'].astype('category').cat.codes
```

### Grid Search for each solver

```
In [19]: predictors = ['alpha', 'delta', 'g', 'r', 'i', 'redshift', 'plate']

X = df[predictors].values
y = df['class'].values

#Train, Test, Split (Random Seed 42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4

# The other solvers are giving us Convergence warnings. This is another reason we stuck
grid = {'solver':['liblinear', 'sag', 'saga']}
clf_cv = GridSearchCV(LogisticRegression(), grid, cv=5, scoring='accuracy')
clf_cv.fit(X_train, y_train)
print(clf_cv.best_params_)
```

```
C:\Users\Jordan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
    warnings.warn("The max_iter was reached which means "
C:\Users\Jordan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
    warnings.warn("The max_iter was reached which means "
C:\Users\Jordan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
    warnings.warn("The max_iter was reached which means "
C:\Users\Jordan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
    warnings.warn("The max_iter was reached which means "
C:\Users\Jordan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
    warnings.warn("The max_iter was reached which means "
```

```
eWarning: The max_iter was reached which means the coef_ did not converge
    warnings.warn("The max_iter was reached which means "
C:\Users\Jordan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: Convergenc
eWarning: The max_iter was reached which means the coef_ did not converge
    warnings.warn("The max_iter was reached which means "
C:\Users\Jordan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: Convergenc
eWarning: The max_iter was reached which means the coef_ did not converge
    warnings.warn("The max_iter was reached which means "
C:\Users\Jordan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: Convergenc
eWarning: The max_iter was reached which means the coef_ did not converge
    warnings.warn("The max_iter was reached which means "
C:\Users\Jordan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: Convergenc
eWarning: The max_iter was reached which means the coef_ did not converge
    warnings.warn("The max_iter was reached which means "
C:\Users\Jordan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: Convergenc
eWarning: The max_iter was reached which means the coef_ did not converge
    warnings.warn("The max_iter was reached which means "
C:\Users\Jordan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: Convergenc
eWarning: The max_iter was reached which means the coef_ did not converge
    warnings.warn("The max_iter was reached which means "
C:\Users\Jordan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: Convergenc
eWarning: The max_iter was reached which means the coef_ did not converge
    warnings.warn("The max_iter was reached which means "
C:\Users\Jordan\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: Convergenc
eWarning: The max_iter was reached which means the coef_ did not converge
    warnings.warn("The max_iter was reached which means "
{'solver': 'liblinear'}
```

## Drop labels

```
In [20]: df.drop(columns=['class_cat'], inplace=True)
```

## Decision Tree Classifier Exploration

### Testing different hyperparameters

```
In [21]: # we will test max_depth values ranging from 2 to 10
params = {'max_depth':np.arange(2,11)}

# grid search using 5-fold cross validation
clf_cv = GridSearchCV(DecisionTreeClassifier(), params, cv=5)
clf_cv.fit(X_train, y_train)

# print results
print('Best estimator:', clf_cv.best_estimator_)
print('Best accuracy:', clf_cv.best_score_)
```

```
Best estimator: DecisionTreeClassifier(max_depth=9)
Best accuracy: 0.9742871757609851
```

## Testing Models using best parameters from exploration

### Set Predictors and Targets

```
In [22]: # Set predictors and targets
predictors = ['alpha', 'delta', 'g', 'r', 'i', 'redshift', 'plate']
targets = ['GALAXY', 'QSO', 'STAR']
```

```
X = df[predictors].values
y = df[targets].values
```

## Split and Scale Data

In [23]:

```
#Train, Test, Split (Random Seed 42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4)

# Scale X_train and X_test
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

## kNN Classification

In [24]:

```
knn = KNeighborsClassifier(weights='distance', metric='manhattan')
knn.fit(X_train, y_train)

# get predictions
predictions = knn.predict(X_test)

# inverse the one hot encoding
predictions_enc = pd.DataFrame(predictions).idxmax(axis=1)
y_test_enc = pd.DataFrame(y_test).idxmax(axis=1)
# print accuracy
print('Accuracy: ', (predictions_enc == y_test_enc).mean())

# print classification report
print(classification_report(y_test_enc, predictions_enc))

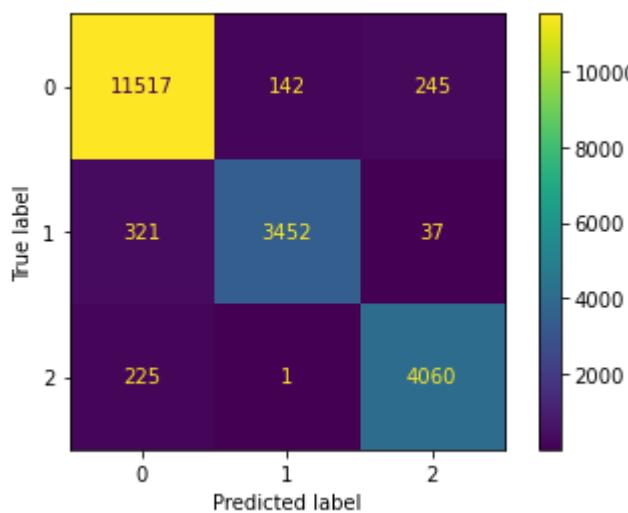
# print confusion matrix
cm = confusion_matrix(y_test_enc, predictions_enc)
print(cm)

# display confusion matrix as heatmap
display = ConfusionMatrixDisplay(cm).plot()
```

Accuracy: 0.95145

	precision	recall	f1-score	support
0	0.95	0.97	0.96	11904
1	0.96	0.91	0.93	3810
2	0.94	0.95	0.94	4286
accuracy			0.95	20000
macro avg	0.95	0.94	0.94	20000
weighted avg	0.95	0.95	0.95	20000

```
[[11517  142  245]
 [ 321  3452   37]
 [ 225      1  4060]]
```



## Decision Tree Classification

In [25]:

```

clf = DecisionTreeClassifier(max_depth=5)
clf.fit(X_train, y_train)

print('Feature Importances: ', clf.feature_importances_)

# get predictions
predictions = clf.predict(X_test)

# inverse one hot encoding
predictions_enc = pd.DataFrame(predictions).idxmax(axis=1)
y_test_enc = pd.DataFrame(y_test).idxmax(axis=1)

# print accuracy
print('Accuracy: ', (predictions_enc == y_test_enc).mean())

# print classification report
print(classification_report(y_test_enc, predictions_enc))

# print confusion matrix
cm = confusion_matrix(y_test_enc, predictions_enc)
print(cm)

# display confusion matrix as heatmap
display = ConfusionMatrixDisplay(cm).plot()

```

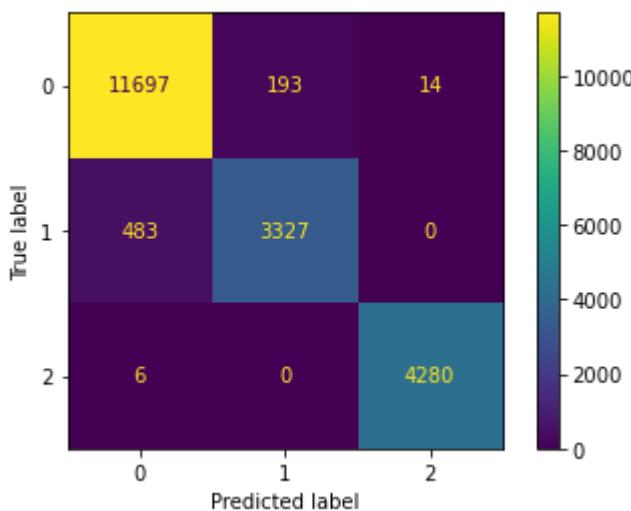
Feature Importances: [0.0000000e+00 0.0000000e+00 4.87853578e-02 0.0000000e+00  
 8.97544233e-04 9.46872377e-01 3.44472091e-03]

Accuracy: 0.9652

	precision	recall	f1-score	support
0	0.96	0.98	0.97	11904
1	0.95	0.87	0.91	3810
2	1.00	1.00	1.00	4286
accuracy				0.97
macro avg	0.97	0.95	0.96	20000
weighted avg	0.96	0.97	0.96	20000

[[11697 193 14]]

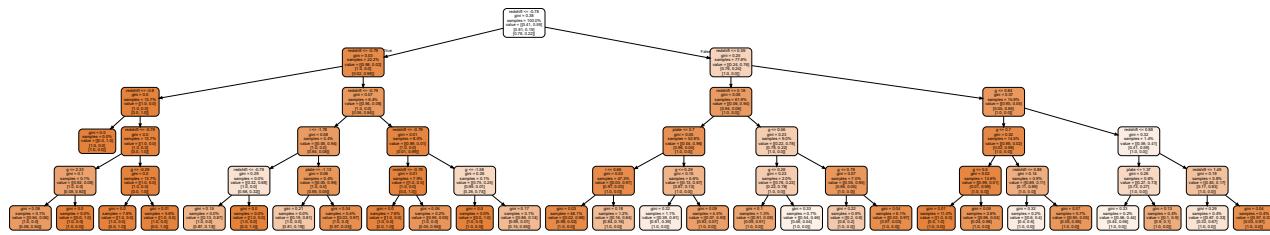
```
[ 483  3327    0]
[    6      0  4280]
```



In [26]:

```
target_names = ['GALAXY', 'QSO', 'STAR']
dot_data = export_graphviz(clf, precision=2, feature_names=predictors, proportion=True,
                           class_names=target_names, filled=True, rounded=True)
graph = graphviz.Source(dot_data)
graph
```

Out[26]:



## SVC

SVC does not require one-hot encoding, so we will instead use the 'class' column as our target.

In [27]:

```
# Set target
target = ['class']

# Set y
y = df[target].values.ravel()

# Train, Test, Split (Random Seed 42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4

# Scale X_train and X_test
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [28]:

```
svc = SVC()
svc.fit(X_train, y_train)

predictions = svc.predict(X_test)
```

```
# print accuracy
print('Accuracy: ', (predictions == y_test).mean())

# print classification report
print(classification_report(y_test, predictions))

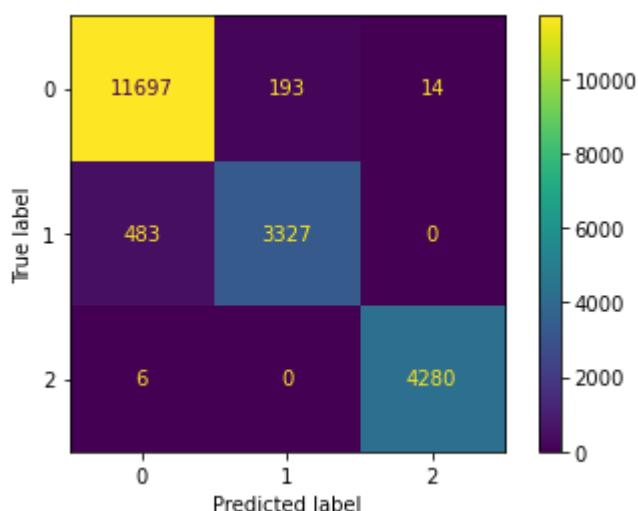
# get confusion matrix
cm = confusion_matrix(y_test_enc, predictions_enc)
print(cm)

# display confusion matrix as heatmap
display = ConfusionMatrixDisplay(cm).plot()
```

Accuracy: 0.9625

	precision	recall	f1-score	support
GALAXY	0.97	0.97	0.97	11904
QSO	0.97	0.91	0.94	3810
STAR	0.95	0.99	0.97	4286
accuracy			0.96	20000
macro avg	0.96	0.96	0.96	20000
weighted avg	0.96	0.96	0.96	20000

`[[11697 193 14]`  
`[ 483 3327 0]`  
`[ 6 0 4280]]`



## Logistic Regression

In [29]:

```
clf = LogisticRegression(solver='liblinear')
clf.fit(X_train, y_train)

predictions = clf.predict(X_test)

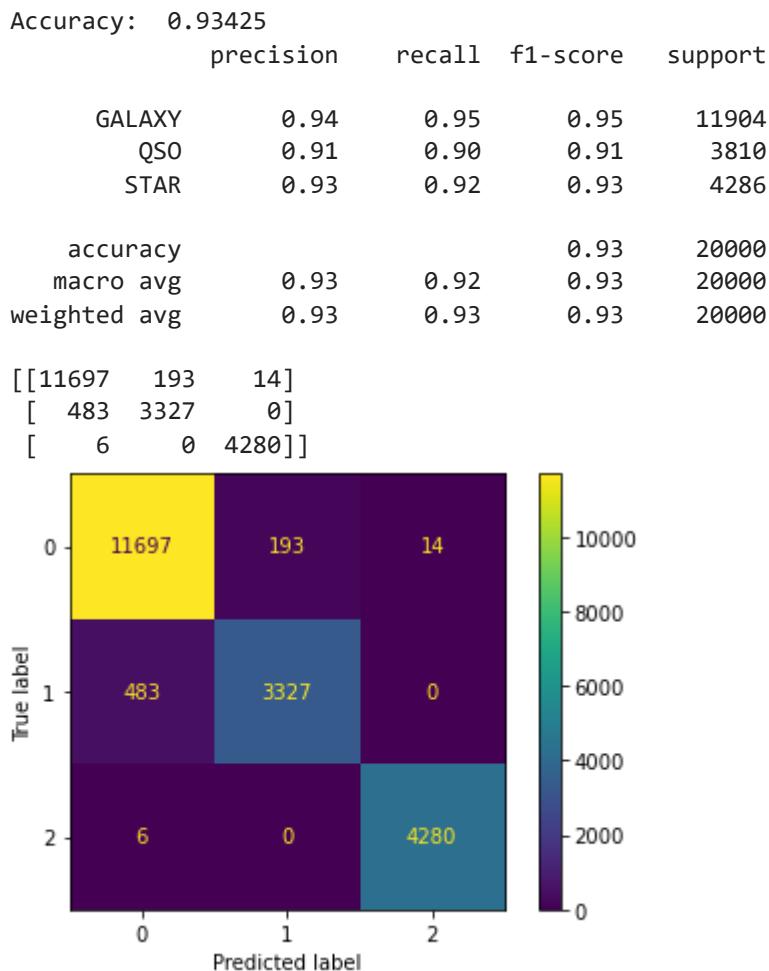
# print accuracy
print('Accuracy: ', (predictions == y_test).mean())

# print classification report
print(classification_report(y_test, predictions))

# get confusion matrix
```

```
cm = confusion_matrix(y_test_enc, predictions_enc)
print(cm)

# display confusion matrix as heatmap
display = ConfusionMatrixDisplay(cm).plot()
```



## Chosen Model

Show decision tree code, graph of hits vs misses and where the majority occurred

## Accuracy

In [30]:

```
#Label encoding
df['class_cat'] = df['class'].astype('category').cat.codes
predictors = ['alpha', 'delta', 'g', 'r', 'i', 'redshift', 'plate']
X = df[predictors]
y = df['class_cat']
#TTS
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4

#default
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
train_predict = clf.predict(X_train)
train_error = (train_predict != y_train).mean()
```

```

test_predict = clf.predict(X_test)
test_error = (test_predict != y_test).mean()
train_accuracy = 1 - train_error
test_accuracy = 1 - test_error

print("Training accuracy:", str("{:.2f}").format(100*train_accuracy)) + "%")
print("Test accuracy:", str("{:.2f}").format(100*test_accuracy)) + "%")

```

Training accuracy: 100.00%  
Test accuracy: 96.32%

## Misclassifications

Per the previous section's confusion matrix, it can be seen that the majority of misclassifications are GALAXY/QSO misclassifications. Furthermore, these misclassifications tend to happen in a specific redshift and g range. Let's take a look at them

In [31]:

```

#Data
fig, ax = plt.subplots(figsize = (30,15))
redshift_range_upper = 1.7
redshift_range_lower = 0.05
g_range_upper = 23.5
g_range_lower = 18.5
# df_small = df[(df['redshift'] < redshift_range_upper) & (df['redshift'] > redshift_ra
df_small = df[(df['redshift'] < redshift_range_upper) & (df['redshift'] > redshift_rang
object_counts = df_small['class'].value_counts()
object_percentages = 100*(df_small['class'].value_counts() / df['class'].value_counts())
sns.scatterplot(x='redshift', y='g', hue='class', data=df_small, ax=ax)
handles, labels = ax.get_legend_handles_labels()
labels = [ 'GALAXY: ' + str(object_counts['GALAXY']) + ' (' + str(round(object_percenta
    'QSO: ' + str(object_counts['QSO'])) + ' (' + str(round(object_percentages['QS
ax.legend(handles, labels, loc='lower left')

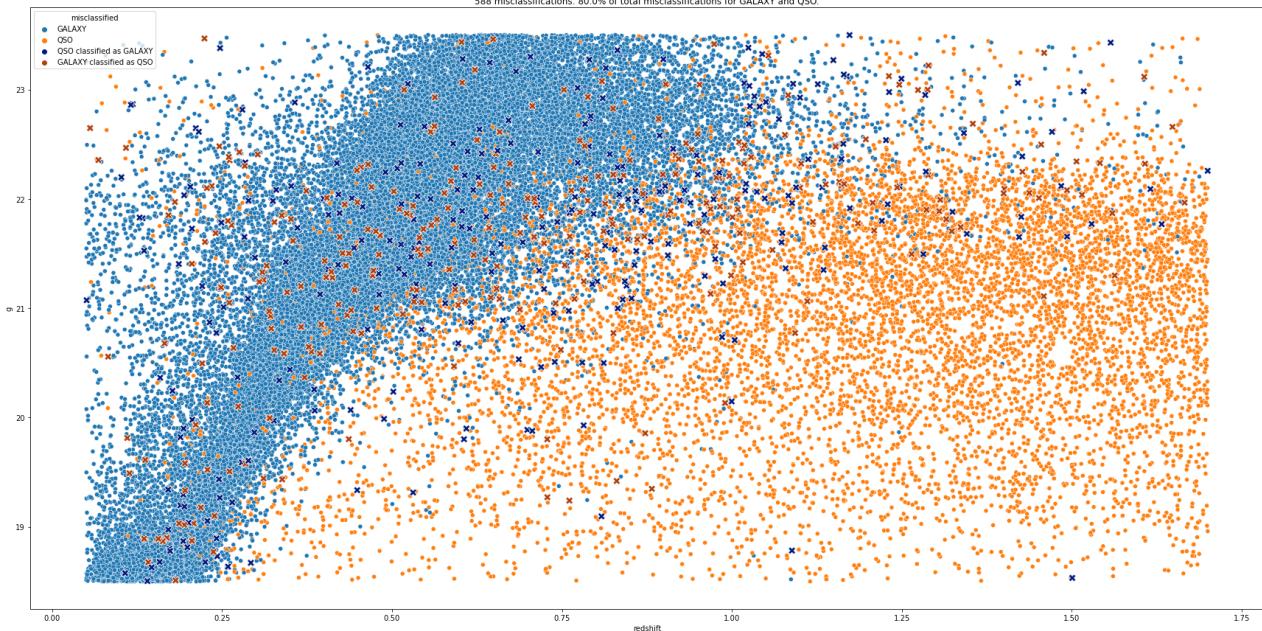
#Misclassifications
misclassified_train = X_train[train_predict != y_train]
misclassified_test = X_test[test_predict != y_test]
shared_indices = []
for index in misclassified_test.index.values:
    if (index in df_small.index):
        shared_indices.append(index)
misclassified_in_df_small = df_small.loc[shared_indices]
misclassified_in_df_small['misclassified'] = misclassified_in_df_small['class'].apply(1
sns.scatterplot(x='redshift', y='g', hue='misclassified', data=misclassified_in_df_smal

ax.set_title('Redshift vs G classes, redshift between ' + str(redshift_range_lower) + ' '
# ax.set_title('Redshift vs G classes, redshift between ' + str(redshift_range_lower) +
print("\nTotal misclassified:", misclassified_test.shape[0])
print("Total misclassified in redshift range:[", str(redshift_range_lower), str(redshif
#Drop Labels
df.drop(columns=['class_cat'], inplace=True)

```

Total misclassified: 736  
Total misclassified in redshift range:[ 0.05 1.7 ]: 588  
GALAXY classified as QSO 301  
QSO classified as GALAXY 287  
Name: misclassified, dtype: int64

Redshift vs G classes, redshift between 0.05 and 1.7, g between 18.5 and 23.5.  
588 misclassifications. 80.0% of total misclassifications for GALAXY and QSO.



## Bonus: Our model in action

### Import SDSS data set from previous year (2016)

We have to be wary of overlap between our set and the set from the former year. Let's clean up the columns and get rid of the duplicates

In [32]:

```
#2016 SDSS data
# had to use local storage for this file because it was too big to upload to Github. Th
df_sdss_2016 = pd.read_csv(r"C:\Users\Jordan\Downloads\archive\sdss.csv")

#Clean up data and match columns to main data
df_sdss_2016.rename(columns={'object_id': 'obj_ID', 'right_ascension': 'alpha', 'declin
    'g_magnitude': 'g', 'r_magnitude': 'r', 'i_magnitude': 'i', 'z_magn
    'obs_run_number': 'run_ID', 'rerun_number': 'rerun_ID', 'camera_
    'field_number': 'field_ID', 'spectro_object_id': 'spec_obj_ID',
    'observation_date': 'MJD', 'fiber_id': 'fiber_ID'}, inplace=True

#drop duplicates in 2016 set
df_sdss_2016.drop_duplicates(subset=['obj_ID'], inplace=True)

#reread the original dataframe to get obj_ID back
df_og = pd.read_csv("https://raw.githubusercontent.com/Deelane/CST383---Final/main/star

#find overlapping rows in main set and 2016 set
obj_id_set = set(df_sdss_2016['obj_ID'])
ids_to_drop = []
for obj_id in df_og['obj_ID']:
    if obj_id in obj_id_set:
        ids_to_drop.append(obj_id)

#remove overlapping rows between main set and 2016 set
df_sdss_2016 = df_sdss_2016[df_sdss_2016['obj_ID'].isin(ids_to_drop) == False]

#drop non-used columns
```

```
df_sdss_2016.drop(columns=['obj_ID', 'run_ID', 'rerun_ID', 'field_ID', 'fiber_ID', 'spec  
#drop bad data  
bad_indices = df_sdss_2016[(df_sdss_2016['i'] == -9999)].index  
df_sdss_2016.drop(bad_indices, inplace=True) #drop the bad data  
df_sdss_2016.reset_index(drop=True, inplace=True)
```

## Testing our model against these 732,646 new objects

In [33]:

```
#Label encoding  
df_sdss_2016['class_cat'] = df_sdss_2016['class'].astype('category').cat.codes  
predictors = ['alpha', 'delta', 'g', 'r', 'i', 'redshift', 'plate']  
X = df_sdss_2016[predictors].values  
y = df_sdss_2016['class_cat'].values  
  
#Using our already trained model to predict all 732,646 new objects  
test_predict = clf.predict(X)  
test_error = (test_predict != y).mean()  
test_accuracy = 1 - test_error  
print("Number of objects predicted:", len(test_predict))  
print("Accuracy:", str("{:.2f}".format(100*test_accuracy)) + "%")
```

Number of objects predicted: 732646

Accuracy: 98.29%