1 Lab 14: Flavours of Boosting and Feature Importance

In this notebook, we build a photometric redshift estimator using various boosting methods: AdaBoost and various flavors of Gradient Boosted Trees (GBM, HistGBM, and XGBoost). We also look at using RandomizedSearchCV in order to improve our exploration of parameter space.

Our goal is to estimate photometric redshifts starting from observations of galaxy magnitudes in six different photometric bands (u, g, r, i, z, y).

In [31]:
```
import numpy as np
import pandas as pd
from scipy import stats
import matplotlib
import matplotlib.pyplot as plt
from joblib import parallel_backend

pd.set_option('display.max_columns', 100)
pd.set_option('display.max_rows', 100)
pd.set_option('display.max_colwidth', 100)

font = {'size'   : 16}
matplotlib.rc('font', **font)
matplotlib.rc('xtick', labelsize=14)
matplotlib.rc('ytick', labelsize=14)
#matplotlib.rcParams.update({'figure.autolayout': True})
matplotlib.rcParams['figure.dpi'] = 100
```

In [32]:
```
from sklearn import metrics
from sklearn.model_selection import cross_validate, cross_val_score, KFold, cross_val_predict, GridSearchCV,RandomizedSearchCV
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier, AdaBoostRegressor, GradientBoostingRegressor,RandomForestRegressor

# read in pre-processed data from Lab 13
sel_features = pd.read_csv('sel_features.csv', sep = '\t')
sel_target = pd.read_csv('sel_target.csv')
```

1.1 Step 1: AdaBoost again

In Lab 13, we showed that for AdaBoostRegressor, stacking learners that are too weak doesn't help. Let's do a more thorough optimization of hyperparameters for AdaBoost.

First, look at the model hyperparameters using model.get_params. As estimator use DecisionTreeRegressor(). Set up a **GridSearchCV** and find the best model within the suggested parameters.

'estimator__max_depth':[6,10,None], 'loss':['linear','square'], 'n_estimators':[20,50,100], 'learning_rate': [0.3,0.5,1.0]

You can use 3-fold CV. Use n_jobs=-1 to request all resources. If the run time turns out to be too long on your machine, use **RandomizedSearchCV** with n_iter=50 or 100 instead. Normally we would only use the randomized search in higher dimensional parameter space, but it should be sufficient here. You can increase n_inter according to your patience threshold!

What are the test scores, optimal parameters and outlier fraction of the winning model?

In [33]:
```
model = AdaBoostRegressor()
model.get_params()
```

Out[33]:
```
{'base_estimator': 'deprecated',
 'estimator': None,
 'learning_rate': 1.0,
 'loss': 'linear',
 'n_estimators': 50,
 'random_state': None}
```

In [34]:
```
model = AdaBoostRegressor(estimator=DecisionTreeRegressor())
params = {'estimator__max_depth':[6,10,None], 'loss':['linear','square'], 'n_estimators':[20,50,100], 'learning_rate': [0.3,0.5,1.0]}
```

In [35]:
```
grid_search = RandomizedSearchCV(model, params, cv=3,n_jobs=-1)  # Adjust cv as needed
grid_search.fit(sel_features,sel_target)
```

```
C:\Users\kesha\miniconda3\envs\cs425\lib\site-packages\sklearn\utils\validation.py:1184: DataConversionWarning: A column-vector y was passed
when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[35]:
```
RandomizedSearchCV(cv=3,
                   estimator=AdaBoostRegressor(estimator=DecisionTreeRegressor()),
                   n_jobs=-1,
                   param_distributions={'estimator__max_depth': [6, 10, None],
                                        'learning_rate': [0.3, 0.5, 1.0],
                                        'loss': ['linear', 'square'],
                                        'n_estimators': [20, 50, 100]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [36]:
```
print(f"""

The optimal parameters are {grid_search.best_params_} ,

The Test scores are the following: {grid_search.best_score_}
""")
```

```
The optimal parameters are {'n_estimators': 50, 'loss': 'square', 'learning_rate': 0.3, 'estimator__max_depth': None} ,

The Test scores are the following: 0.7315660837031187
```

1.2 Step 2: Gradient Boosted Trees

Let's repeat the analysis above for three GB variants:

1. GradientBoostingRegressor

Suggested parameters: 'max_depth':[6,10,None], 'loss':['squared_error','absolute_error'], 'n_estimators':[20,50,100], 'learning_rate': [0.1,0.3,0.5]

2. HistGradientBoostingRegressor

HistGradientBoostingRegressor (inspired by LightGBM) works by binning the features into integer-valued bins (the default value is 256, but this parameter can be adjusted; note however that 256 is the maximum!), which greatly reduces the number of splitting points to consider, and results in a vast reduction of computation time, especially for large data sets.

Suggested paramters: 'max_depth':[6,10,None], 'loss':['squared_error','absolute_error'], 'max_iter':[20,50,100], 'learning_rate': [0.1,0.3,0.5]

3. XGBRegressor. The latter requires installation of the python package xgboost.

XGBoost stands for “Extreme Gradient Boosting”. It is sometimes known as "regularized" GBM, as it has a default regularization term on the weights of the ensemble, and is more robust to overfitting. It has more flexibility in defining weak learners, as well as the objective (loss) function (note that this doesn't apply to the base estimators, e.g. how splits in trees are chosen, but on the loss that is used to compute pseudoresiduals and gradients).

Suggested parameters: 'max_depth':[6,10,None], 'n_estimators':[50,100,200], 'learning_rate': [0.1, 0.3,0.5], 'objective':['reg:squarederror','reg:squaredlogerror']

What are the test scores, optimal parameters and outlier fraction for the winning models in each case?

In [37]:
```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid_gbr = {
    'max_depth': [6, 10, None],
    'loss': ['squared_error', 'absolute_error'],
    'n_estimators': [20, 50, 100],
    'learning_rate': [0.1, 0.3, 0.5]
}

# Initialize the model
gbr = GradientBoostingRegressor()

# Set up the grid search
grid_search_gbr = GridSearchCV(estimator=gbr, param_grid=param_grid_gbr, cv=3, n_jobs=-1, scoring='neg_mean_squared_error')

# Fit the model (assuming X_train, y_train are your features and target variable)
grid_search_gbr.fit(sel_features, sel_target)

# Best parameters and score
print("Best parameters:", grid_search_gbr.best_params_)
print("Best score:", -grid_search_gbr.best_score_)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[37], line 19
     16 grid_search_gbr = GridSearchCV(estimator=gbr, param_grid=param_grid_gbr, cv=3, n_jobs=-1, scoring='neg_mean_squared_error')
     18 # Fit the model (assuming X_train, y_train are your features and target variable)
---> 19 grid_search_gbr.fit(sel_features, sel_target)
     21 # Best parameters and score
     22 print("Best parameters:", grid_search_gbr.best_params_)

File ~\miniconda3\envs\cs425\lib\site-packages\sklearn\base.py:1151, in _fit_context.<locals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)
   1144     estimator._validate_params()
   1146 with config_context(
   1147     skip_parameter_validation=(
   1148         prefer_skip_nested_validation or global_skip_validation
   1149     )
   1150 ):
-> 1151     return fit_method(estimator, *args, **kwargs)
```

In [ ]:
```
from sklearn.experimental import enable_hist_gradient_boosting  # noqa
from sklearn.ensemble import HistGradientBoostingRegressor

# Define the parameter grid
param_grid_hgbr = {
    'max_depth': [6, 10, None],
    'loss': ['squared_error', 'absolute_error'],
    'max_iter': [20, 50, 100],
    'learning_rate': [0.1, 0.3, 0.5]
}

# Initialize the model
hgbr = HistGradientBoostingRegressor()

# Set up the grid search
grid_search_hgbr = GridSearchCV(estimator=hgbr, param_grid=param_grid_hgbr, cv=3, n_jobs=-1, scoring='neg_mean_squared_error')

# Fit the model
grid_search_hgbr.fit(sel_features, sel_target)

# Best parameters and score
print("Best parameters:", grid_search_hgbr.best_params_)
print("Best score:", -grid_search_hgbr.best_score_)
```

In [ ]:
```
from xgboost import XGBRegressor

# Define the parameter grid
param_grid_xgb = {
    'max_depth': [6, 10, None],
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.1, 0.3, 0.5],
    'objective': ['reg:squarederror', 'reg:squaredlogerror']
}

# Initialize the model
xgb = XGBRegressor()

# Set up the grid search
grid_search_xgb = GridSearchCV(estimator=xgb, param_grid=param_grid_xgb, cv=3,n_jobs=-1, scoring='neg_mean_squared_error')

# Fit the model
grid_search_xgb.fit(sel_target, sel_features)

# Best parameters and score
print("Best parameters:", grid_search_xgb.best_params_)
print("Best score:", -grid_search_xgb.best_score_)
```

1.3 Step 3: Feature importance

Although one of the main goals of ML is to be able to make predictions as accurate as possible, it is equally desirable to gain insight into why the machine makes certain decisions. To this end, we would like to be able to identify those features that are most important in the classification or regression process. This is sometimes called "interpretable machine learning".

Our new ensemble methods all have a property .feature_importances_. What does this property give you in the case of a random forest?

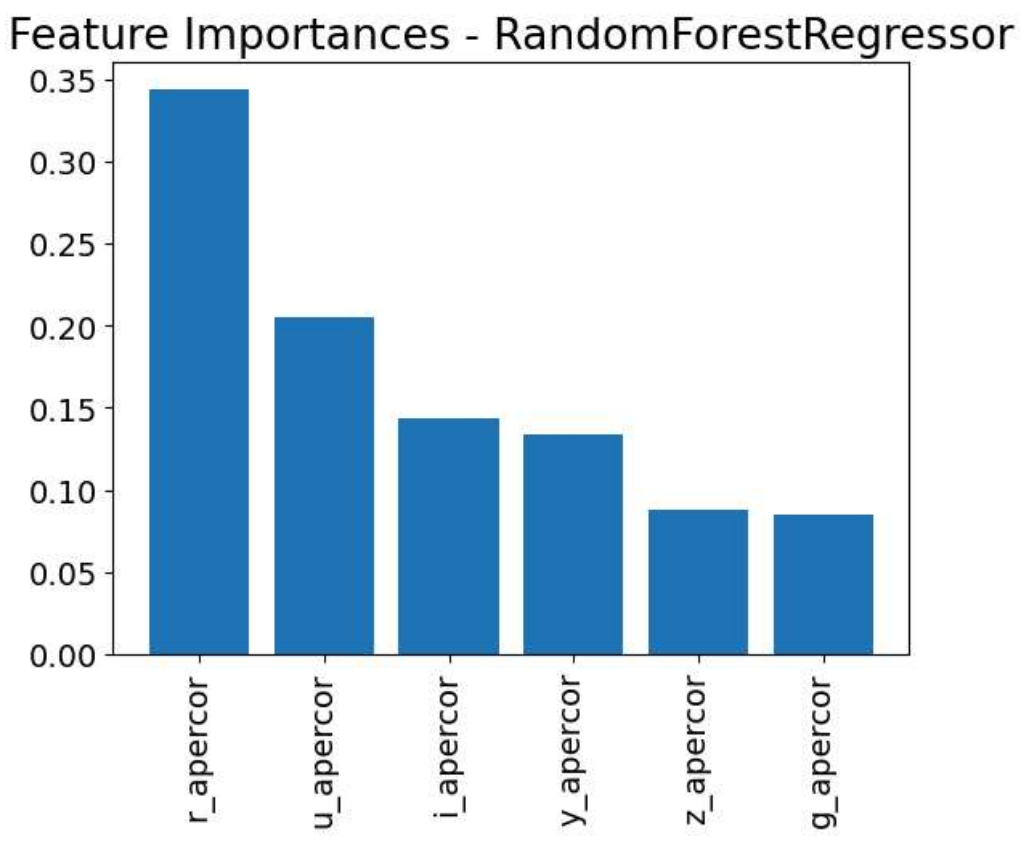
For the RandomForestRegressor, the AdaBoostRegressor, and the XGBRegressor, find those importances, rank them in order of importance and make a bar chart for each regressor that plots the importances of the six features in our redshift dataset, starting with the most important one. To this end, just fit the models to the full dataset, no split test/train needed. You can use the optimized parameters you found above. Do you get the same answer from all three regressors?

In [38]:
```
# Fit RandomForestRegressor
rf = RandomForestRegressor()  # Replace optimized_params_rf with the found optimal parameters
rf.fit(sel_features, sel_target)
rf_importances = rf.feature_importances_

# Create a function to plot feature importances
def plot_feature_importances(importances, title):
    indices = np.argsort(importances)[::-1]
    plt.figure()
    plt.title(title)
    plt.bar(range(sel_features.shape[1]), importances[indices])
    plt.xticks(range(sel_features.shape[1]), sel_features.columns[indices], rotation=90)
    plt.show()

# Plotting the feature importances
plot_feature_importances(rf_importances, 'Feature Importances - RandomForestRegressor')
```

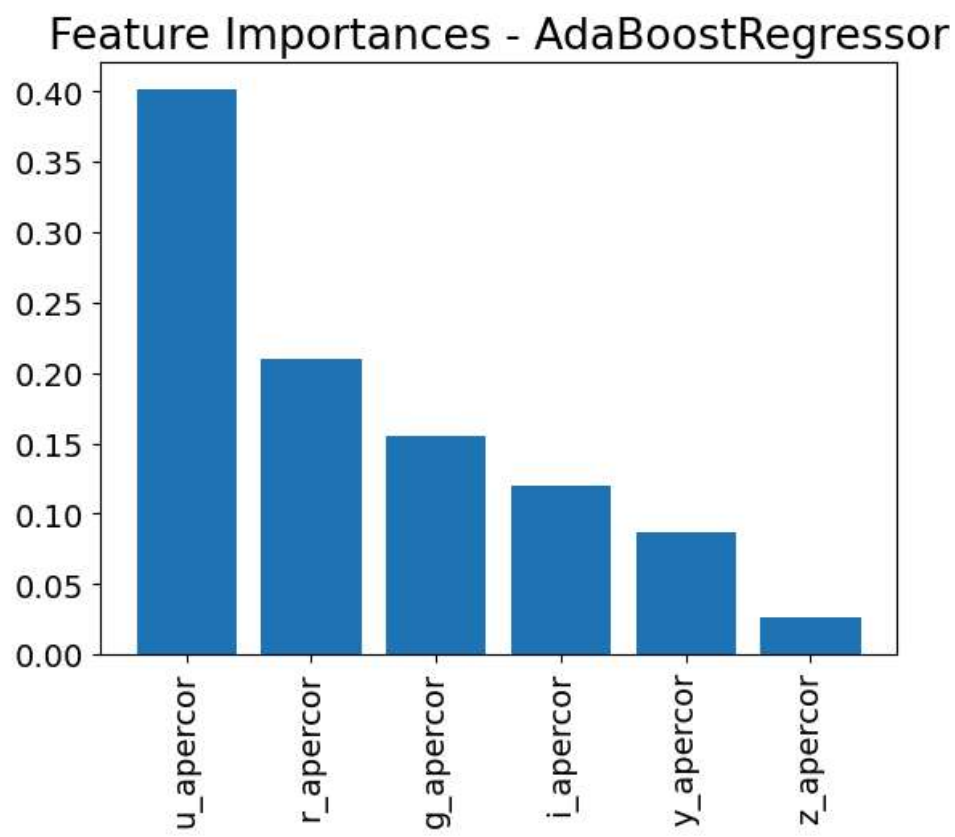
```
C:\Users\kesha\miniconda3\envs\cs425\lib\site-packages\sklearn\base.py:1151: DataConversionWarning: A column-vector y was passed when a 1d a
rray was expected. Please change the shape of y to (n_samples,), for example using ravel().
  return fit_method(estimator, *args, **kwargs)
```



In [39]:
```
# Fit AdaBoostRegressor
ada = AdaBoostRegressor()  # Replace optimized_params_ada with the found optimal parameters
ada.fit(sel_features, sel_target)
ada_importances = ada.feature_importances_

plot_feature_importances(ada_importances, 'Feature Importances - AdaBoostRegressor')
```

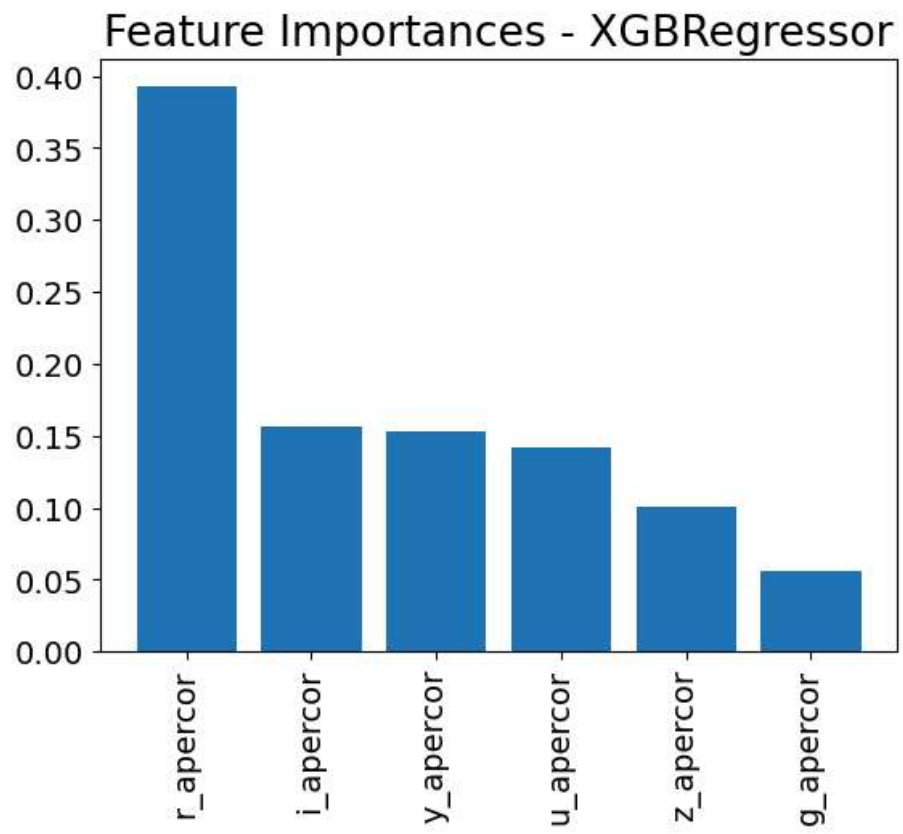
```
C:\Users\kesha\miniconda3\envs\cs425\lib\site-packages\sklearn\utils\validation.py:1184: DataConversionWarning: A column-vector y was passed
when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```



In [42]:
```
from xgboost import XGBRegressor
# Fit XGBRegressor
xgb = XGBRegressor()  # Replace optimized_params_xgb with the found optimal parameters
xgb.fit(sel_features, sel_target)
xgb_importances = xgb.feature_importances_

plot_feature_importances(xgb_importances, 'Feature Importances - XGBRegressor')
```

```
C:\Users\kesha\miniconda3\envs\cs425\lib\site-packages\xgboost\data.py:299: FutureWarning: is_sparse is deprecated and will be removed in a
future version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if is_sparse(dtype):
C:\Users\kesha\miniconda3\envs\cs425\lib\site-packages\xgboost\data.py:301: FutureWarning: is_categorical_dtype is deprecated and will be re
moved in a future version. Use isinstance(dtype, CategoricalDtype) instead
  elif is_categorical_dtype(dtype) and enable_categorical:
C:\Users\kesha\miniconda3\envs\cs425\lib\site-packages\xgboost\data.py:332: FutureWarning: is_categorical_dtype is deprecated and will be re
moved in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if is_categorical_dtype(dtype)
C:\Users\kesha\miniconda3\envs\cs425\lib\site-packages\xgboost\data.py:323: FutureWarning: is_categorical_dtype is deprecated and will be re
moved in a future version. Use isinstance(dtype, CategoricalDtype) instead
  return is_int or is_bool or is_float or is_categorical_dtype(dtype)
C:\Users\kesha\miniconda3\envs\cs425\lib\site-packages\xgboost\data.py:332: FutureWarning: is_categorical_dtype is deprecated and will be re
moved in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if is_categorical_dtype(dtype)
C:\Users\kesha\miniconda3\envs\cs425\lib\site-packages\xgboost\data.py:323: FutureWarning: is_categorical_dtype is deprecated and will be re
moved in a future version. Use isinstance(dtype, CategoricalDtype) instead
  return is_int or is_bool or is_float or is_categorical_dtype(dtype)
```



1.4 Step 4: Correlations among features

It is very common for the features to be correlated amonst each other. This can complicate the interpretation of feature importance, since two highly correlated features might receive equal weight. Correlated features also introduce reduncancy in the overall model so it is desirable to develop strategies to remove them.

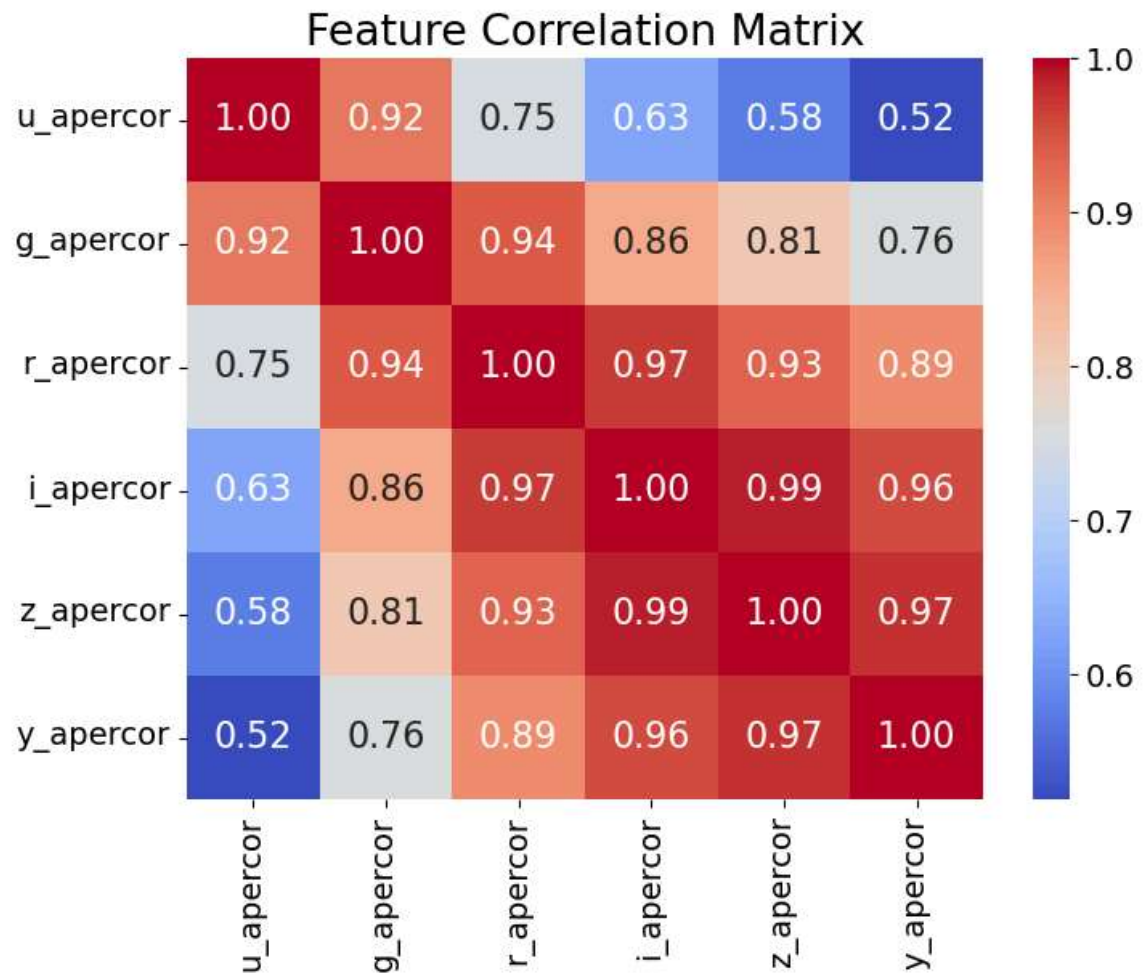
1. Determine the cross-correlations between the six features in our dataset. There are several options to do this, in pandas we could just use the .corr() method. Find the pair of features that is most correlated.

In [49]:
```
import seaborn as sns

correlation_matrix = sel_features.corr()

# Find the two features with the highest correlation (ignoring the diagonal)
# We use numpy to create the mask for the diagonal
mask = np.eye(correlation_matrix.shape[0], dtype=bool)
max_corr = correlation_matrix.mask(mask).abs().stack().idxmax()

plt.figure(figsize=(8, 6))
heatmap = sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar=True)
plt.title('Feature Correlation Matrix')
plt.show()
```



In [53]:
```
print("""
Answer:
----------------------------------------------------------------
The features that are most correlated are the i and z features
""")
```

```
Answer:
----------------------------------------------------------------
The features that are most correlated are the i and z features
```

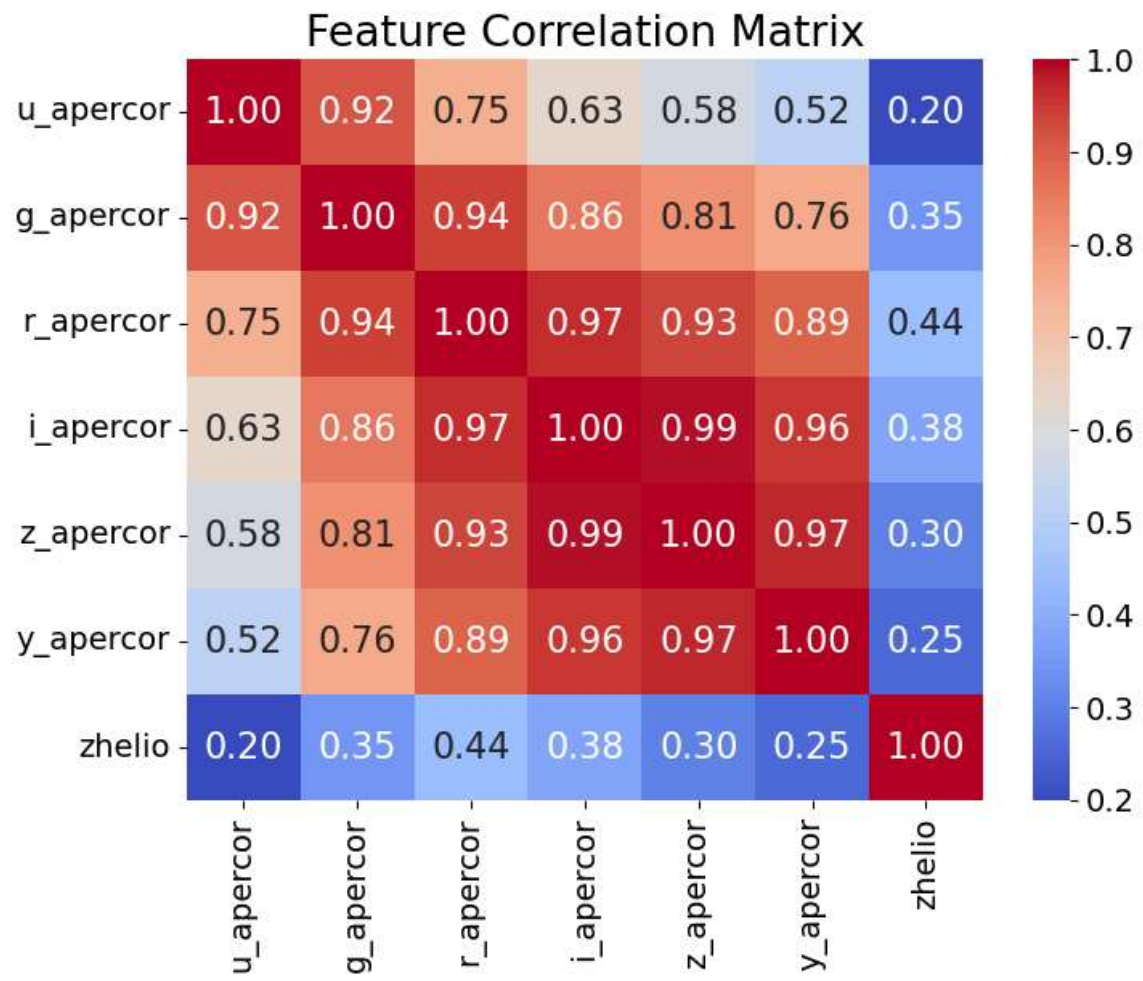
2. From these two features, which one is less correlated with the target variable? Eliminate that feature and redo the feature ranking for the three regressors with this new model that now only has 5 features. Also compute a cross-validated score for these smaller models, are they worse than for the full model? What do you think?

In [50]:
```
import seaborn as sns

correlation_matrix = pd.DataFrame.join(sel_features,sel_target).corr()

# Find the two features with the highest correlation (ignoring the diagonal)
# We use numpy to create the mask for the diagonal
mask = np.eye(correlation_matrix.shape[0], dtype=bool)
max_corr = correlation_matrix.mask(mask).abs().stack().idxmax()

plt.figure(figsize=(8, 6))
heatmap = sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar=True)
plt.title('Feature Correlation Matrix')
plt.show()
```



In [55]:
```
print("""
Answer:
-------------------------------------------------
We opt to drop the the z variable as compared to the i feature, as the i feature is more closely related to the target variable
""")
```

```
Answer:
-------------------------------------------------
We opt to drop the the z variable as compared to the i feature, as the i feature is more closely related to the target variable
```

In [51]:
```
df = sel_features.drop('z_apercor', axis=1)
```

In [56]:
```
grid_search.fit(df,sel_target)
print(f"""

The optimal parameters are {grid_search.best_params_} ,

The Test scores are the following: {grid_search.best_score_}
""")
```

```
C:\Users\kesha\miniconda3\envs\cs425\lib\site-packages\sklearn\utils\validation.py:1184: DataConversionWarning: A column-vector y was passed
when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)


The optimal parameters are {'n_estimators': 20, 'loss': 'square', 'learning_rate': 0.5, 'estimator__max_depth': None} ,

The Test scores are the following: 0.7087852317268691
```