HW3

November 1, 2020

0.1 Homework 3:

0.1.1 For this assignment, I tested many models and feature engineering techniques. In the first part, I show how I selected the features/genes and how I can further manipulate the data with dimensionality reduction. In the second part, I added information from external datasets, specifically the RACS and Cell Line datasets. In the third part, I developing a scoring to rank drug sensitivity for each cell line.

0.2 Part 1a:

0.2.1 Load and preprocess the data.

[46]:
```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn.feature_selection import VarianceThreshold
from sklearn.decomposition import PCA
from sklearn.svm import SVR
from sklearn.linear_model import Ridge
from sklearn.linear_model import ElasticNet
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.ensemble import RandomForestRegressor
%matplotlib inline

data_dir = './MLiC-Homework'
```

[48]:
```
# Import the full gene expression data
gene_exp_df = pd.read_csv(os.path.join(data_dir,␣
↪'Cell_line_RMA_proc_basalExp_transposed.tsv'), header = 0,index_col=0,␣
↪skiprows=[1], sep='\t')
gene_exp_df.index.rename('COSMIC_ID', inplace=True)
gene_exp_df.index = gene_exp_df.index.map(lambda x: x.split('.')[1])
```

```python
gene_exp_df.index = gene_exp_df.index.astype('int64')

full_gene = gene_exp_df
full_gene.head()
```

[48]:

```
              TSPAN6      TNMD       DPM1     SCYL3  C1orf112       FGR  \
COSMIC_ID
906826      7.632023  2.964585  10.379553  3.614794  3.380681  3.324692
687983      7.548671  2.777716  11.807341  4.066887  3.732485  3.152404
910927      8.712338  2.643508   9.880733  3.956230  3.236620  3.241246
1240138     7.797142  2.817923   9.883471  4.063701  3.558414  3.101247
1240139     7.729268  2.957739  10.418840  4.341500  3.840373  3.001802

                 CFH     FUCA2      GCLC      NFYA  …  LINC00526      PPY2  \
COSMIC_ID                                          …
906826      3.566350  8.204530  5.235118  5.369039  …   6.786925  2.997054
687983      7.827172  6.616972  5.809264  7.209653  …   5.317911  3.263745
910927      2.931034  8.191246  5.426841  5.120747  …   3.143006  3.112145
1240138     7.211707  8.630643  5.617714  4.996434  …   3.153896  3.151576
1240139     3.375422  8.296950  5.669418  4.180205  …   3.652660  2.918475

           Unnamed: 17730  Unnamed: 17731  KRT18P55  Unnamed: 17733  POLRMTP1  \
COSMIC_ID
906826           3.109774        7.882377  3.331134        2.852537  3.130696
687983           3.059424        8.681302  2.992611        2.776771  3.260982
910927           2.930254        8.707886  2.886574        2.685307  3.176239
1240138          2.850726        7.872535  3.812119        3.436412  3.074432
1240139          2.849537        8.945953  3.412586        2.951270  3.213545

             UBL5P2  TBC1D3P5  Unnamed: 17737
COSMIC_ID
906826      9.986616  3.073724        7.284733
687983      9.002814  3.000182        8.504804
910927      9.113243  2.916274        7.059092
1240138     9.958284  3.256500        7.318125
1240139     9.938978  3.396126        7.726867

[5 rows x 17737 columns]
```

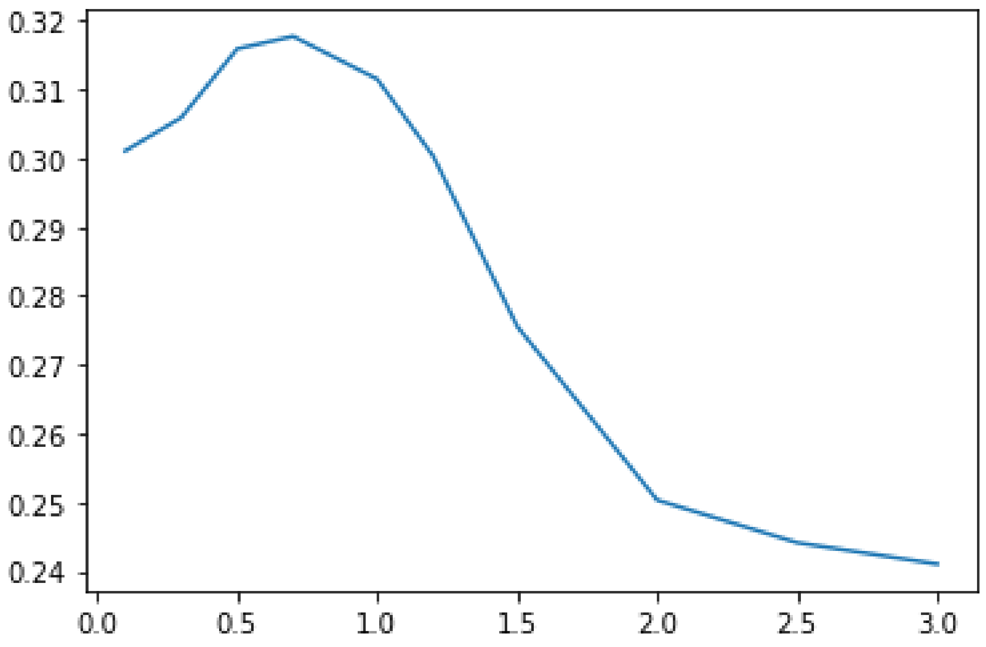
0.2.2 In the given normalized dataset there are 17737 genes. Many of these genes are uninformative because there is very little variance across cell lines. Thus, we need to determine a variance threshold such that genes with a variance lower than the threshold are removed from the data. For the purposes of the assignment, we test multiple variances and see how they affect model performance. I use a simple linear model, ridge regression as the model.

[51]:
```
from sklearn.feature_selection import VarianceThreshold
scoring = 'r2'
model = Ridge
param_grid = {'alpha':np.logspace(3,6,15)}

drugs = list(ic50_df[ic50_df.COSMIC_ID == 924100].DRUG_ID.sample(n=10,␣
↪random_state =0))
variances = {}
for drug in drugs:
    print(f'Drug: {drug}')
    small_ic50 = ic50_df[ic50_df.DRUG_ID == drug]
    small_merged = small_ic50.merge(gene_exp_df, left_on = 'COSMIC_ID',␣
↪right_on = 'COSMIC_ID', how='left')
    small_merged.drop('COSMIC_ID', axis=1, inplace=True)
    small_merged.dropna(axis=0, how='any',inplace=True)
    y = small_merged.LN_IC50
    X = small_merged.iloc[:,1:]
    for var in [0.1, 0.3, 0.5, 0.7, 1.0, 1.2, 1.5, 2.0, 2.5, 3.0]:


        sel = VarianceThreshold(threshold = var)
        X = sel.fit_transform(X)

        X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = .2,␣
↪random_state=0)
        clf = GridSearchCV(model(), param_grid, scoring=scoring, cv=5,␣
↪n_jobs=-1)
        clf.fit(X_train, y_train)
        if var in variances:
            variances[var].append(clf.best_score_)
        else:
            variances[var]=[clf.best_score_]


for k,v in variances.items():
    print(f'Features below variance of {k} removed: {np.mean(v)}')

datapts = list(zip(*(variances.values())))

np.var(datapts, axis=1)
```

```
datapts = datapts[:3]+datapts[5:]

plt.plot(variances.keys(), np.mean(datapts, axis=0))
```

```
Drug: 171
Drug: 1005
Drug: 290
Drug: 235
Drug: 305
Drug: 263
Drug: 1194
Drug: 345
Drug: 1375
Drug: 1037
Features below variance of 0.1 removed: 0.330338359755036
Features below variance of 0.3 removed: 0.3373802648186358
Features below variance of 0.5 removed: 0.35019882531931257
Features below variance of 0.7 removed: 0.3563095492960514
Features below variance of 1.0 removed: 0.3581216648930875
Features below variance of 1.2 removed: 0.35271980606819947
Features below variance of 1.5 removed: 0.33814701023735005
Features below variance of 2.0 removed: 0.3264947421222291
Features below variance of 2.5 removed: 0.329346947912169
Features below variance of 3.0 removed: 0.29640077519523167
```

```
[51]: [<matplotlib.lines.Line2D at 0x7fafec59ea00>]
```



0.2.3 From this analysis, the graph begins to plateau around 0.5, so we will use this as the variance threshold moving forward. The number of features/genes reduces from 17737 to 6136.

[52]:
```
# Import the IC50 data. These will be the labels for the samples
ic50_df = pd.read_csv(os.path.join(data_dir, 'v17_fitted_dose_response.csv'),␣
↪usecols = ['COSMIC_ID', 'DRUG_ID', 'LN_IC50'], dtype={'COSMIC_ID':np.int64})
ic50_df.head()
```

[52]:
```
   COSMIC_ID  DRUG_ID  LN_IC50
0     924100     1026     0.72
1     924100     1028     2.66
2     924100     1029     3.34
3     924100     1030     5.16
4     924100     1031    -4.33
```

[53]:
```
def merge(threshold, gene_data):
    '''Merge gene data (features) to the IC50 data (labels)'''
    if threshold == 0:
        x = gene_data
    else:
        x = gene_data.loc[:,~(gene_data.var()<threshold)]
    merged = ic50_df.merge(x, left_on = 'COSMIC_ID', right_on = 'COSMIC_ID',␣
↪how='left')
    merged.drop('COSMIC_ID', axis=1, inplace=True)
    merged.dropna(axis=0, how='any',inplace=True)
    return merged
```

[ ]:
```
# Dataframe with both labels and features (validation threshold of 0.5)
xs_merged = merge(0.5, full_gene)
```

[277]:
```
xs_merged.shape
```

[277]:
```
(213650, 6136)
```

[55]:
```
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.metrics import accuracy_score
from sklearn.pipeline import Pipeline


def fully_train_and_evaluate(scoring, data, model, param_grid, drugs):
    '''Fine-tuning, training, and evaluating all-in-one'''
    test_scores = {}
    train_scores = {}
    best_params = {}
    clf_test_scores = {}
    for i, drug in enumerate(drugs):
#         print(f'Drug {i+1}, Drug id {drug}')
        test_scores[drug] = []
        train_scores[drug] = []
        best_params[drug] = []
        clf_test_scores[drug] = []

        d = data[data.DRUG_ID == drug].drop('DRUG_ID', axis = 1).
 ↪reset_index(drop =True)
        X = d.iloc[:,1:]
        y = d.LN_IC50
        X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = .2,␣
 ↪random_state=0)

        clf = GridSearchCV(model, param_grid, scoring='r2', cv=5, n_jobs=-1)
        clf.fit(X_train, y_train)

        best_model = clf.best_estimator_
        y_pred = best_model.predict(X_test)
#         print(clf.best_score_)
        train_scores[drug]=clf.best_score_
        test_scores[drug]=best_model.score(X_test, y_test)

        mean = y.mean()
        std = y.std()

        def label(value):
            if value < mean - std:
                return 's'
            elif value > mean + std:
                return 'r'
            else:
                return 'i'

        clf_y_pred = list(map(label, y_pred))
        clf_y_test = list(map(label, y_test))
        clf_test_scores[drug]=(accuracy_score(clf_y_test, clf_y_pred))

    return train_scores, test_scores, clf_test_scores
```

0.2.4 Note: For the classification portion of this part of the assignment, I determine thresholds for each class (Sensitive, Intermediate, Resistant). Then, I use these threshold to map the regression prediction to a class. The thresholds are determined by the mean and standard deviation of the IC50 values. If the IC50 value is less than one standard deviation below the mean, the cell line is sensitive to the drug. If the IC50 value is more than one standard deviation above the mean, the cell line is resistant. Otherwise, the cell line is considered Intermediate.

0.3 Part 1b:

0.3.1 In order to determine which model to use for this problem, I tested many models, including Ridge Regression, Support Vector Regression, ElasticNet, RandomForestRegression, and MLPRegressor; however, time and time again, the Ridge Regression model performed better on average. Below I show the test for SVR and Ridge Regression.

```
[58]: drugs = list(ic50_df.DRUG_ID.unique()[:10])
      models = {'svr':(SVR(), {'C':[1, 10], 'epsilon': [0.1, 0.2, 0.4, 0.6, 0.8, 1]}),
                'ridge': (Ridge(), {'alpha': np.logspace(3, 6, 15)})}
      # models = {'ridge': (Ridge(), {'alpha': np.logspace(3, 6, 15)})}

      train = {}
      reg_test = {}
      clf_test = {}

      for j, model in models.items():
          train_scores, test_scores, clf_test_scores = fully_train_and_evaluate('r2',␣
       ↪xs_merged, model[0], model[1], drugs)
          train[j] = train_scores
          reg_test[j] = test_scores
          clf_test[j] = clf_test_scores
```

```
[66]: svr_test = np.mean(list(reg_test['svr'].values()))
      r_test = np.mean(list(reg_test['ridge'].values()))
      svr_clf = np.mean(list(clf_test['svr'].values()))
      r_clf = np.mean(list(clf_test['ridge'].values()))

      print(f"On average across the drug models, the r2 value for SVR is {svr_test}")
      print(f"On average across the drug models, the r2 value for Ridge is {r_test}")

      print(f"On average across the drug models, the accuracy for SVR is {svr_clf}")
      print(f"On average across the drug models, the accuracy for Ridge is {r_clf}")
```

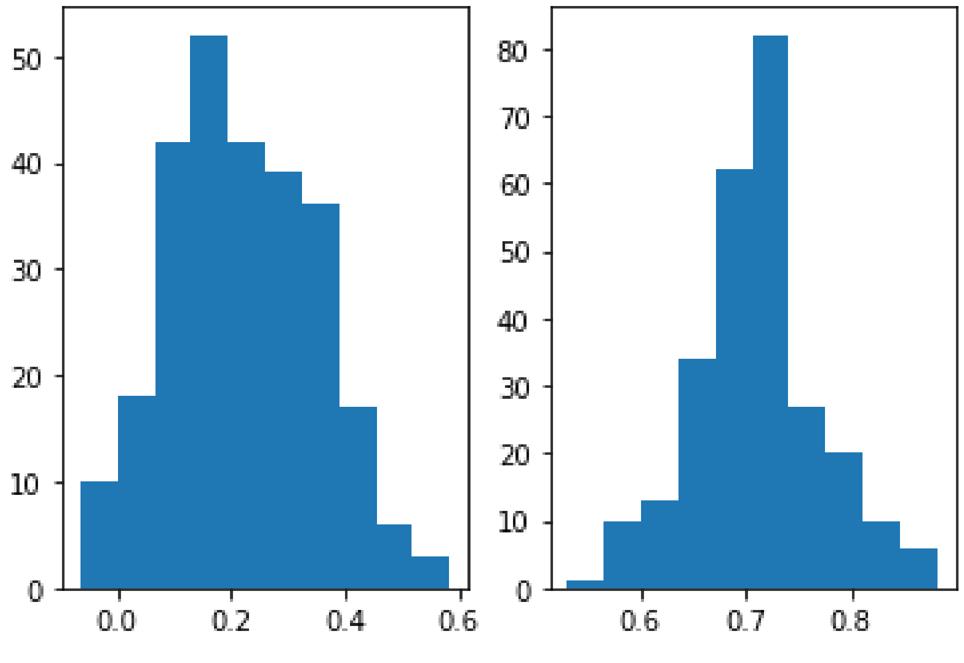
```
On average across the drug models, the r2 value for SVR is 0.26346361045007477
On average across the drug models, the r2 value for Ridge is 0.27947337796140204
On average across the drug models, the accuracy for SVR is 0.7206548347613219
On average across the drug models, the accuracy for Ridge is 0.722405820753434
```

0.3.2 We observe that Ridge performs slightly better than SVR

0.3.3 We will use Ridge regression and for each drug model, we will do a grid search to find the optimal alpha for the model.

[67]:
```
drugs = list(ic50_df.DRUG_ID.unique())
model=(Ridge(), {'alpha': np.logspace(3, 6, 15)})

xs_train, xs_test, xs_clf = fully_train_and_evaluate('r2', xs_merged, model[0],␣
↪model[1], drugs)
```

[76]:
```
# pd.DataFrame(list(xs_test.values())).hist()
def plot_histograms(lst1, lst2):
    plt.subplot(1, 2, 1)  # 1 line, 2 rows, index nr 1 (first position in the␣
↪subplot)
    plt.hist(lst1)
    plt.subplot(1, 2, 2)  # 1 line, 2 rows, index nr 2 (second position in the␣
↪subplot)
    plt.hist(lst2)
    plt.show()

plot_histograms(list(xs_test.values()), list(xs_clf.values()))
```



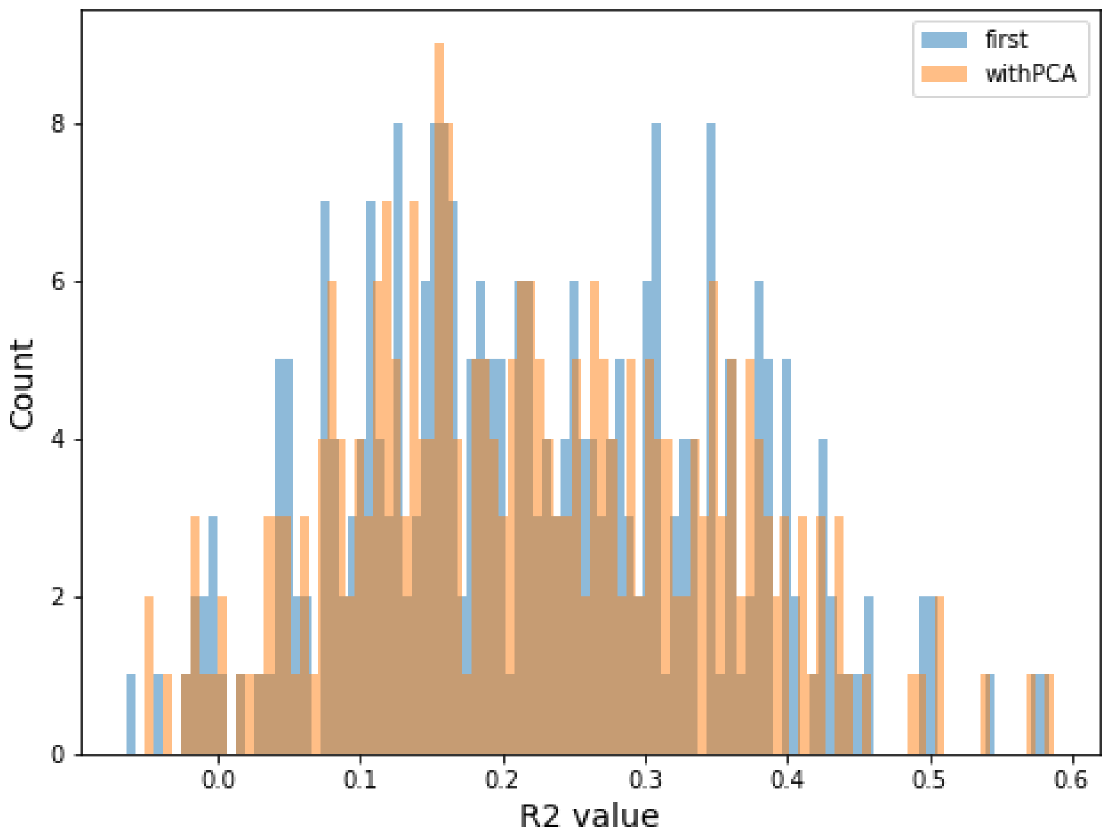
0.3.4 Above is a distribution of the evaluation metrics across all drug models. Left is for the regression metric r2. Right is accuracy (for the classification module we integrated into our train and evaluate function.

0.3.5 Sometimes the model performance can be improved with dimensionality reduction on the input features. PCA is a dimensionality reduction technique which does a linear mapping of the data to a lower dimension in a way that maximizes the variance of the data.

```
[74]: xs_gene = full_gene.loc[:,~(full_gene.var()<0.5)]
```

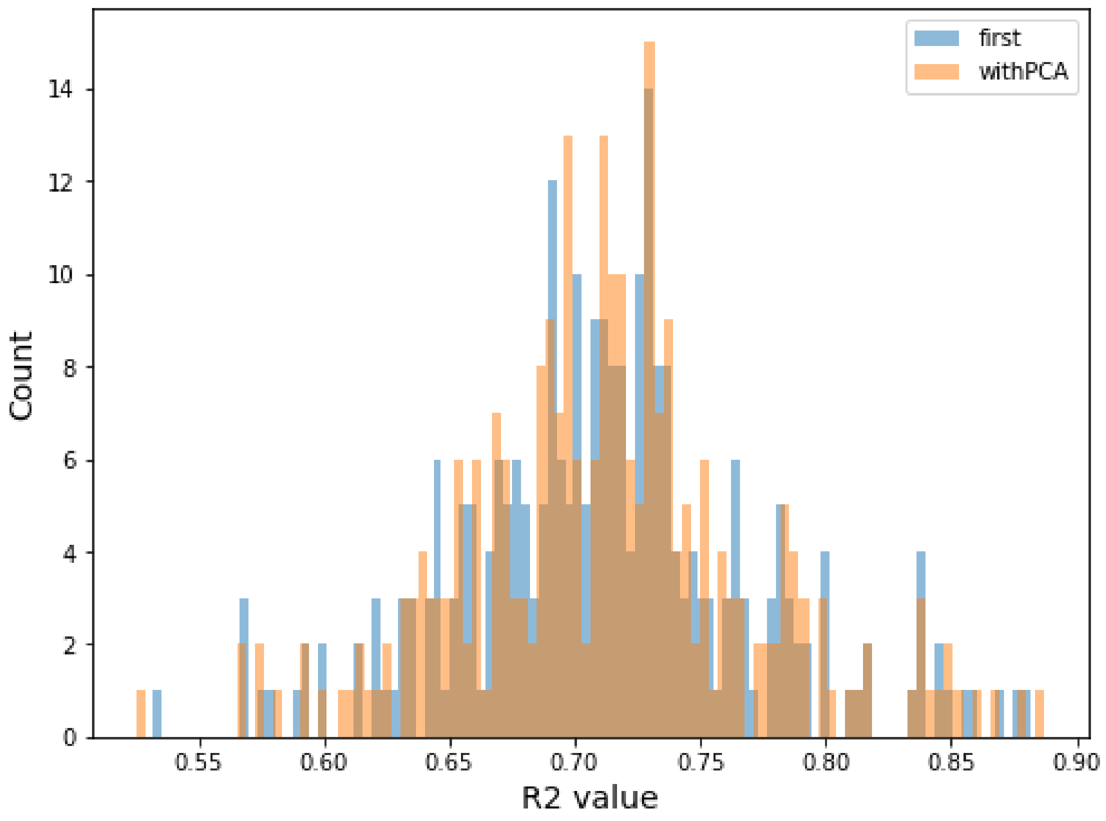
```
[75]: pca = PCA(n_components=500)
      xs_pca = pca.fit_transform(X=xs_gene)



      xs_pca = pd.DataFrame(xs_pca)
      xs_pca.index = gene_exp_df.index

      xs_pca_merged = merge(0, xs_pca)
      drugs = list(ic50_df.DRUG_ID.unique())
      model=(Ridge(), {'alpha': np.logspace(3, 6, 15)})
      pcatrain_scores, pcatest_scores, pcaclf_test_scores =␣
       ↪fully_train_and_evaluate('r2', xs_pca_merged, model[0], model[1], drugs)
```

```
[85]: # plot_histograms(list(pcatest_scores.values()), list(pcaclf_test_scores.
       ↪values()))

      plt.figure(figsize=(8,6))
      plt.hist(list(xs_test.values()), bins=100, alpha=0.5, label="first")
      plt.hist(list(pcatest_scores.values()), bins=100, alpha=0.5, label="withPCA")
      plt.xlabel("R2 value", size=14)
      plt.ylabel("Count", size=14)
      plt.legend(loc='upper right')
      plt.show()
      print(f'Average r2 across all 265 drug models: {np.mean(list(xs_test.
       ↪values()))}')
      print(f'Average r2 across all 265 drug models with reduced data: {np.
       ↪mean(list(pcatest_scores.values()))}')
```



```
Average r2 across all 265 drug models: 0.22266750866444052
Average r2 across all 265 drug models with reduced data: 0.22012428738797846
```

[88]:
```
# plot_histograms(list(pcatest_scores.values()), list(pcaclf_test_scores.
 ↪values()))

plt.figure(figsize=(8,6))
plt.hist(list(xs_clf.values()), bins=100, alpha=0.5, label="first")
plt.hist(list(pcaclf_test_scores.values()), bins=100, alpha=0.5,␣
 ↪label="withPCA")
plt.xlabel("R2 value", size=14)
plt.ylabel("Count", size=14)
plt.legend(loc='upper right')
plt.show()
print(f'Average accuracy across all 265 drug models: {np.mean(list(xs_clf.
 ↪values()))}')
print(f'Average accuracy across all 265 drug models with reduced data: {np.
 ↪mean(list(pcaclf_test_scores.values()))}')
```



```
Average accuracy across all 265 drug models: 0.7121669420813148
Average accuracy across all 265 drug models with reduced data:
0.7124761061082282
```

[97]:
```
def print_top_bottom_10(lst, top=True):
    sorted_drug_models = sorted(lst, key=lambda x: x[1], reverse=top)
    print(sorted_drug_models[:10])


print("Top 10 models based on r2 metric\n")
print_top_bottom_10(list(xs_test.items()), top=True)
print()
print("Top 10 models based on accuracy metric\n")
print_top_bottom_10(list(xs_clf.items()), top=True)
print()

print("Bottom 10 models based on r2 metric\n")
print_top_bottom_10(list(xs_test.items()), top=False)
print()

print("Bottom 10 models based on accuracy metric\n")
```

```
print_top_bottom_10(list(xs_clf.items()), top=False)
```

```
Top 10 models based on r2 metric

[(1373, 0.5840322481392634), (1526, 0.5772350772539481), (1372,
0.5404766874560114), (1008, 0.5057634943313833), (275, 0.5043811688712867),
(226, 0.4988718246207241), (253, 0.49352782175453147), (1498,
0.4579355911707542), (1378, 0.4576136303459901), (271, 0.45139074404615553)]

Top 10 models based on accuracy metric

[(277, 0.8817204301075269), (34, 0.8780487804878049), (312, 0.8702702702702703),
(268, 0.8603351955307262), (1242, 0.8540540540540541), (226, 0.85), (1373,
0.8465909090909091), (281, 0.8432432432432433), (1013, 0.8395061728395061),
(1377, 0.8378378378378378)]

Bottom 10 models based on r2 metric

[(62, -0.06475938863413644), (1527, -0.043674042999427565), (110,
-0.021957766407385337), (86, -0.01568182913825522), (207,
-0.014233154751329913), (1042, -0.010443180490896431), (254,
-0.01043519984136787), (166, -0.006365078101229882), (53, -0.00607791433796856),
(6, -0.0005787113781152708)]

Bottom 10 models based on accuracy metric

[(176, 0.5314285714285715), (64, 0.5679012345679012), (89, 0.5679012345679012),
(208, 0.5698924731182796), (41, 0.575), (87, 0.5802469135802469), (344,
0.5891891891891892), (302, 0.592391304347826), (134, 0.5932203389830508), (1248,
0.5989010989010989)]
```

[99]:
```
print("Top 10 models based on r2 metric\n")
print_top_bottom_10(list(pcatest_scores.items()), top=True)
print()
print("Top 10 models based on accuracy metric\n")
print_top_bottom_10(list(pcaclf_test_scores.items()), top=True)
print()

print("Bottom 10 models based on r2 metric\n")
print_top_bottom_10(list(pcatest_scores.items()), top=False)
print()

print("Bottom 10 models based on accuracy metric\n")
print_top_bottom_10(list(pcaclf_test_scores.items()), top=False)
```

```
Top 10 models based on r2 metric

[(1373, 0.5869354105412552), (1526, 0.5730332948556274), (1372,
0.5385317772894929), (275, 0.5092682109977129), (1008, 0.50463123686106), (226,
0.49738465569188206), (253, 0.48602442650374955), (1498, 0.4551466426158254),
(271, 0.4415192368875138), (1378, 0.4390788826549844)]

Top 10 models based on accuracy metric

[(277, 0.8870967741935484), (34, 0.8780487804878049), (268, 0.8659217877094972),
(312, 0.8594594594594595), (1242, 0.8540540540540541), (226, 0.85), (1377,
0.8486486486486486), (1373, 0.8465909090909091), (281, 0.8432432432432433),
(1013, 0.8395061728395061)]

Bottom 10 models based on r2 metric

[(1527, -0.051142141574277655), (62, -0.04954753581725213), (254,
-0.035100644307628714), (110, -0.02040578863334508), (1042,
-0.0162254366461696), (86, -0.014356512772685903), (207, -0.012875518689364984),
(166, -0.01020719980662399), (53, -0.0018917614273319394), (6,
0.002593830781742912)]

Bottom 10 models based on accuracy metric

[(176, 0.5257142857142857), (64, 0.5679012345679012), (89, 0.5679012345679012),
(41, 0.575), (208, 0.5752688172043011), (87, 0.5802469135802469), (302,
0.592391304347826), (134, 0.5932203389830508), (1248, 0.5989010989010989), (344,
0.6054054054054054)]
```

0.4 Part 1 summary:

0.4.1 I analyzed both with and without PCA. The dimensionality reduction in general did not improve or worsen the model performance. For the top/bottom 10 models results, even if the order was not exactly the same, the top/bottom results were as a whole in agreement between the models trained on the original data and those trained on the data with PCA. Even if PCA did not tremendously improve the performances of all models, it allows us to train with less features. We continue to part 2 with the PCA data.

0.5 Part 2:

0.5.1 In this part, I added additional information to the data that was transformed by PCA. I reasoned that information about tissue type could help the model, especially if tissue type influences what cell lines are sensitive to certain drugs.

```
[100]: racs = pd.read_csv(os.path.join(data_dir, 'RACS_in_cell_lines.csv'))
       cells = pd.read_csv(os.path.join(data_dir, 'Cell_Lines_Details.csv'),␣
        ↪usecols=['COSMIC identifier', 'GDSC\rTissue\rdescriptor 2'])
```

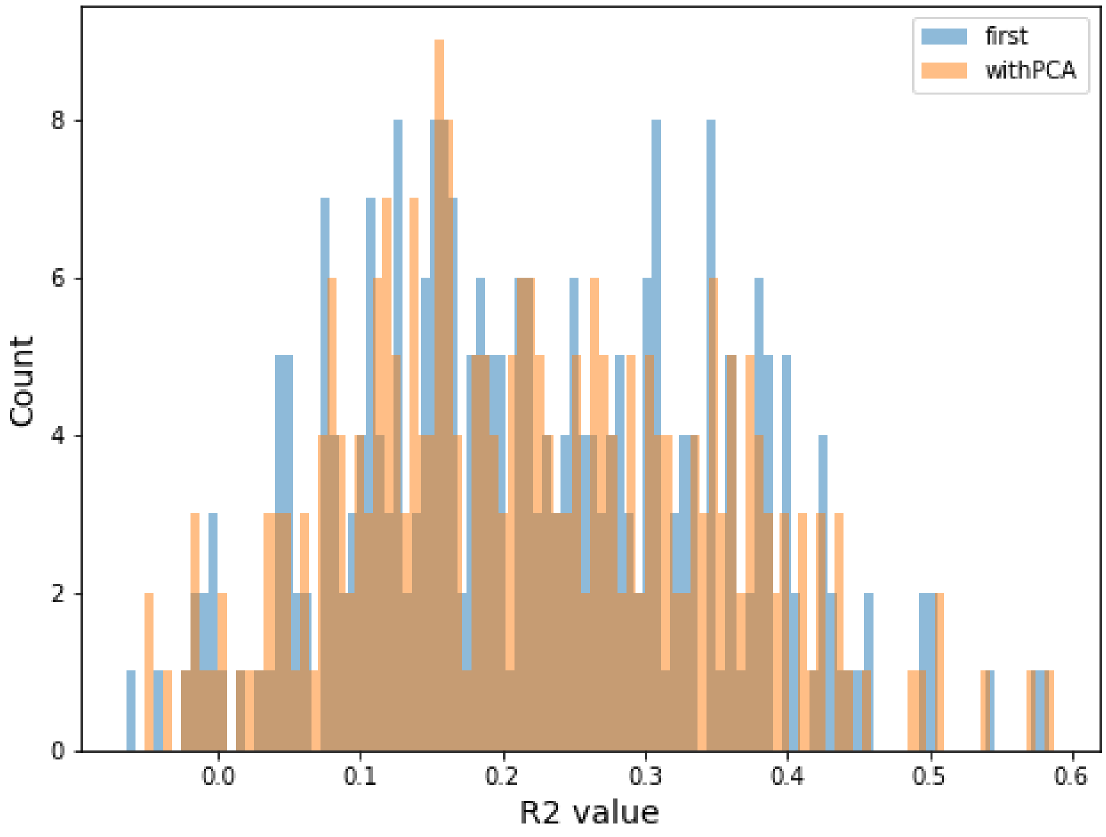
```
[101]: cells.rename({'COSMIC identifier': 'COSMIC_ID'}, axis = 1, inplace =True)
       cells = cells.merge(racs, left_on = 'COSMIC_ID', right_on = 'COSMIC_ID',␣
        ↪how='left')
       cells['Tissue identifier 2'].fillna(cells['GDSC\rTissue\rdescriptor 2'],␣
        ↪inplace =True)
       cells  = cells[['COSMIC_ID', 'Tissue identifier 2']]
       cells.drop_duplicates(inplace=True)
       cells = cells.iloc[:-1].set_index('COSMIC_ID')
       cells.head()
```

```
[101]:           Tissue identifier 2
       COSMIC_ID
       906794.0         head and neck
       753531.0         head and neck
       753532.0         head and neck
       753535.0         head and neck
       1290724.0        head and neck
```

```
[102]: tissues = pd.get_dummies(cells['Tissue identifier 2'])
```

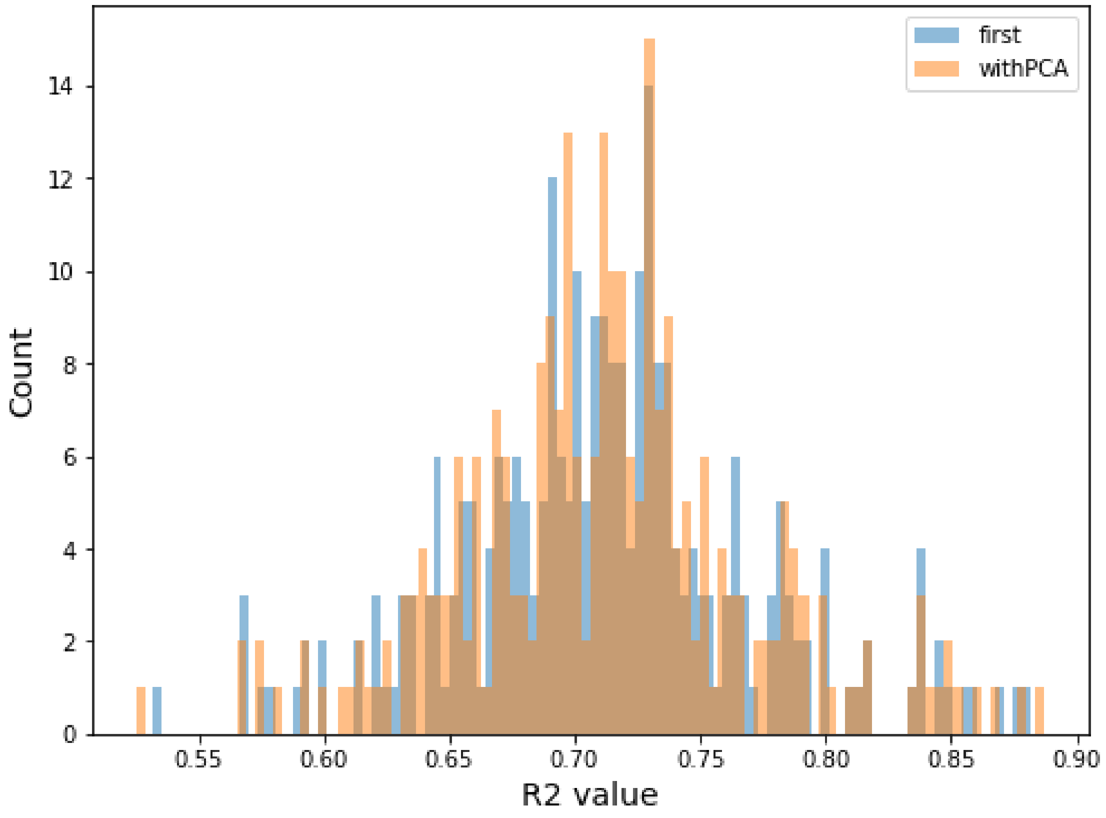
```
[104]: pca = PCA(n_components=500)
       xs_pca = pca.fit_transform(X=xs_gene)
       xs_pca = pd.DataFrame(xs_pca)
       xs_pca.index = gene_exp_df.index

       PCA_with_tissue_merged = xs_pca.merge(tissues, left_on='COSMIC_ID',␣
        ↪right_on='COSMIC_ID', how='inner')
       PCA_with_tissue_merged = merge(0, PCA_with_tissue_merged)

       drugs = list(ic50_df.DRUG_ID.unique())
       # models = {'svr':(SVR(), {'C':[1, 10], 'epsilon': [0.1, 0.2, 0.4, 0.6, 0.8,␣
        ↪1]}),
       #           'ridge': (Ridge(), {'alpha': np.logspace(3, 6, 15)})}
       model=(Ridge(), {'alpha': np.logspace(3, 6, 15)})
       pca_tissue_train, pca_tissue_test, pca_tissue_clf =␣
        ↪fully_train_and_evaluate('r2', PCA_with_tissue_merged, model[0], model[1],␣
        ↪drugs)
```

```
[106]: # plot_histograms(list(pcatest_scores.values()), list(pcaclf_test_scores.
        ↪values()))

       plt.figure(figsize=(8,6))
       plt.hist(list(xs_test.values()), bins=100, alpha=0.5, label="first")
       plt.hist(list(pcatest_scores.values()), bins=100, alpha=0.5, label="withPCA")
       plt.xlabel("R2 value", size=14)
       plt.ylabel("Count", size=14)
       plt.legend(loc='upper right')
       plt.show()
       print(f'Average r2 across all 265 drug models with reduced data: {np.
       ↪mean(list(pcatest_scores.values()))}')
       print(f'Average r2 across all 265 drug models with reduced data and tissue␣
       ↪information: {np.mean(list(pca_tissue_test.values()))}')
```



```
Average r2 across all 265 drug models with reduced data: 0.22012428738797846
Average r2 across all 265 drug models with reduced data and tissue information:
0.2157272555651142
```

[107]:
```
# plot_histograms(list(pcatest_scores.values()), list(pcaclf_test_scores.
↪values()))

plt.figure(figsize=(8,6))
plt.hist(list(xs_clf.values()), bins=100, alpha=0.5, label="first")
plt.hist(list(pcaclf_test_scores.values()), bins=100, alpha=0.5,␣
↪label="withPCA")
plt.xlabel("R2 value", size=14)
plt.ylabel("Count", size=14)
plt.legend(loc='upper right')
plt.show()
print(f'Average accuracy across all 265 drug models with reduced data: {np.
 ↪mean(list(pcaclf_test_scores.values()))}')
print(f'Average accuracy across all 265 drug models with reduced data and␣
 ↪tissue information: {np.mean(list(pca_tissue_clf.values()))}')
```



```
Average accuracy across all 265 drug models with reduced data:
0.7124761061082282
Average accuracy across all 265 drug models with reduced data and tissue
information: 0.7140045104816958
```

0.6 Part 2 summary:

0.6.1 I added the tissue information to the processed data from part 1. The tissue information was added as a one hot encoding to the pca-transformed data. Although the average r2, calculated across all drug models, did not improve with the additional tissue information, the classification accuracy slightly improved.

0.7 Part 3a: Develop a version of your model that can rank order the drugs for a given Cell Line.

I modify my `fully_train_and_evaluate` method so that it returns the best estimator for each drug model. Then I calculate all drug IC50 values for each cell line. With these results, I test a drug sensitivity scoring method to score the cell line drug combinations based on predicted IC50 and the r2 of each drug model. For this method, I sum the IC50 prediction and the r2 value of the model. I do this to push predictions higher in the sensitivity ranking if the model is more performant. However, this ended up being less predictive than just the predicted IC50, so I used the predicted IC50 to get the top 10 drugs (based on sensitivity, smaller IC50 values means better). Then I made a scoring scheme to compare the "true order" with my "predicted order". I iterate the true ordered list and check whether the drug is within 3 places in the predicted order list. This allows some flexibility in the place of the drug within the list.

[128]:
```python
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.metrics import accuracy_score
from sklearn.pipeline import Pipeline

# revamped method so that I can rank order the drugs for each cell line
def fully_train_and_evaluate(scoring, data, model, param_grid, drugs):
    '''Fine-tuning, training, and evaluating all-in-one'''
    test_scores = {}
    train_scores = {}
    best_estimator = {}
    clf_test_scores = {}

    for i, drug in enumerate(drugs):
#         print(f'Drug {i+1}, Drug id {drug}')
        test_scores[drug] = []
        train_scores[drug] = []
        best_estimator[drug] = []
        clf_test_scores[drug] = []

        d = data[data.DRUG_ID == drug].drop('DRUG_ID', axis = 1).
↪reset_index(drop =True)
        X = d.iloc[:,1:]
        y = d.LN_IC50
        X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = .2,␣
↪random_state=0)
```

```
        clf = GridSearchCV(model, param_grid, scoring='r2', cv=5, n_jobs=-1)
        clf.fit(X_train, y_train)

        best_model = clf.best_estimator_
        best_estimator[drug]=best_model
        y_pred = best_model.predict(X_test)
#         print(clf.best_score_)
        train_scores[drug]=clf.best_score_
        test_scores[drug]=best_model.score(X_test, y_test)

        mean = y.mean()
        std = y.std()

        def label(value):
            if value < mean - std:
                return 's'
            elif value > mean + std:
                return 'r'
            else:
                return 'i'

        clf_y_pred = list(map(label, y_pred))
        clf_y_test = list(map(label, y_test))
        clf_test_scores[drug]=(accuracy_score(clf_y_test, clf_y_pred))

    return train_scores, test_scores, clf_test_scores, best_estimator

drugs = list(ic50_df.DRUG_ID.unique())
model=(Ridge(), {'alpha': np.logspace(3, 6, 15)})
pca_tissue_train, pca_tissue_test, pca_tissue_clf, best_estimators =␣
 ↪fully_train_and_evaluate('r2', PCA_with_tissue_merged, model[0], model[1],␣
 ↪drugs)
```

[252]:
```
# Prediction value to sensitivity class mapper
def label(value):
    if value < mean - std:
        return 's'
    elif value > mean + std:
        return 'r'
    else:
        return 'i'
```

[163]:
```
# Dataset to use for getting the predictions. Reminder: From the og data, we␣
 ↪thresholded the variance such that we
# remove those with a variance less than 0.5. Then we performed PCA to reduce␣
 ↪the dimensions to 500 and finally,
# we added the tissue information.
PCA_with_tissue_merged = xs_pca.merge(tissues, left_on='COSMIC_ID',␣
 ↪right_on='COSMIC_ID', how='inner')
PCA_with_tissue_merged = ic50_df.merge(PCA_with_tissue_merged, left_on =␣
 ↪'COSMIC_ID', right_on = 'COSMIC_ID', how='left')
PCA_with_tissue_merged.dropna(axis=0, how='any',inplace=True)
PCA_with_tissue_merged
```

[163]:

```
        COSMIC_ID  DRUG_ID  LN_IC50          0          1          2  \
0          924100     1026     0.72  10.471315  12.832015  35.471705
1          924100     1028     2.66  10.471315  12.832015  35.471705
2          924100     1029     3.34  10.471315  12.832015  35.471705
3          924100     1030     5.16  10.471315  12.832015  35.471705
4          924100     1031    -4.33  10.471315  12.832015  35.471705
…             …       …       …         …          …          …
225395     909907     1053     1.28 -10.339349  31.943581  24.876818
225396     909907     1054     5.48 -10.339349  31.943581  24.876818
225397     909907     1057    -0.14 -10.339349  31.943581  24.876818
225398     909907     1058     1.25 -10.339349  31.943581  24.876818
225399     909907     1059     2.03 -10.339349  31.943581  24.876818

                3         4         5         6  …  pancreas  prostate  \
0       15.701749 -7.623831  7.835479  3.775737  …       0.0       1.0
1       15.701749 -7.623831  7.835479  3.775737  …       0.0       1.0
2       15.701749 -7.623831  7.835479  3.775737  …       0.0       1.0
3       15.701749 -7.623831  7.835479  3.775737  …       0.0       1.0
4       15.701749 -7.623831  7.835479  3.775737  …       0.0       1.0
…             …       …       …       …  …       …
225395  18.519268 -0.249606  1.094017  3.348234  …       0.0       0.0
225396  18.519268 -0.249606  1.094017  3.348234  …       0.0       0.0
225397  18.519268 -0.249606  1.094017  3.348234  …       0.0       0.0
225398  18.519268 -0.249606  1.094017  3.348234  …       0.0       0.0
225399  18.519268 -0.249606  1.094017  3.348234  …       0.0       0.0

        rhabdomyosarcoma  skin_other  soft_tissue_other  stomach  testis  \
0                    0.0         0.0                0.0      0.0     0.0
1                    0.0         0.0                0.0      0.0     0.0
2                    0.0         0.0                0.0      0.0     0.0
3                    0.0         0.0                0.0      0.0     0.0
4                    0.0         0.0                0.0      0.0     0.0
…                  …         …                …      …     …
225395               0.0         0.0                0.0      0.0     0.0
225396               0.0         0.0                0.0      0.0     0.0
225397               0.0         0.0                0.0      0.0     0.0
225398               0.0         0.0                0.0      0.0     0.0
225399               0.0         0.0                0.0      0.0     0.0
```

```
        thyroid  urogenital_system_other  uterus
0           0.0                      0.0     0.0
1           0.0                      0.0     0.0
2           0.0                      0.0     0.0
3           0.0                      0.0     0.0
4           0.0                      0.0     0.0
…         …                      …     …
225395      0.0                      0.0     0.0
225396      0.0                      0.0     0.0
225397      0.0                      0.0     0.0
225398      0.0                      0.0     0.0
225399      0.0                      0.0     0.0

[208998 rows x 558 columns]
```

[276]:
```python
# This gets the ic50 predictions for all drugs for each cell line.

def get_predictions_for_all_combos():
    res = {}
    for i, cosmic_id in enumerate(PCA_with_tissue_merged.COSMIC_ID.unique()):
        cell_line_features = PCA_with_tissue_merged[PCA_with_tissue_merged.
 ↪COSMIC_ID==cosmic_id].iloc[0,3:]
        res[cosmic_id]=[]
        for drug_id, estimator in best_estimators.items():


            if cosmic_id in list(ic50_df[ic50_df.DRUG_ID==drug_id].COSMIC_ID):

                cell_line_true = ic50_df[(ic50_df.COSMIC_ID==cosmic_id) &␣
 ↪(ic50_df.DRUG_ID==drug_id)].LN_IC50.values[0]
                y_pred = estimator.predict([cell_line_features])
                r2_score = pca_tissue_test[drug_id]

                clf_y_pred = label(y_pred)
                clf_y_test = label(cell_line_true)

                acc_score = accuracy_score([clf_y_test], [clf_y_pred])

                res[cosmic_id].append((drug_id, y_pred[0],cell_line_true,␣
 ↪clf_y_pred, clf_y_test, r2_score))
            else:
                pass
    return res

res = get_predictions_for_all_combos()
```

```
[258]: # mapping the prediction to a sensitivity score based on the prediction and the␣
        ↪r2 of the model
       ex = list(map(lambda x: (x[0], x[1]+x[5], x[2], x[3], x[4], x[5]), res[906832]))
       rawpred_based =sorted(res[906832], key=lambda x: x[1])
       adjustedpred_based = sorted(ex, key=lambda x: x[1])
       true_order = sorted(res[906832], key=lambda x: x[2])
```

```
[275]: def score_rank_ordered_lst(truth, rank_ordered_lst):
           similarity = []
           for i, el in enumerate(truth):
               # Checks whether the drug is within three places of the truth ranking.
               start = i-3
               stop = i+3

               if start < 0:
                   z = rank_ordered_lst[:stop]
               elif stop > len(true_order):
                   z = rank_ordered_lst[start:]
               else:
                   z = rank_ordered_lst[start:stop]

               l = [val[0] for val in z]
               if el[0] in l:
                   similarity.append(True)
               else:
                   similarity.append(False)

           return np.mean(similarity)

       print(f'Comparative score for the raw predictions and the truth:␣
        ↪{score_rank_ordered_lst(true_order, rawpred_based)}')
       print(f'Comparative score for the scored predictions and the truth:␣
        ↪{score_rank_ordered_lst(true_order, adjustedpred_based)}')
```

```
Comparative score for the raw predictions and the truth: 0.125
Comparative score for the scored predictions and the truth: 0.12096774193548387
```

0.7.1 Above is an example of the scores for comparing the predicted top 10 list with the ground truth. The comparative score for the adjusted prediction list that depends on the prediction and the r2 of the drug model was typically lower or very similar, so I continued with just the raw predictions for the top 10 lists.

```
[260]: top10 = {}
       truth = {}
       comparative_scores = {}
       for cosmic_id in PCA_with_tissue_merged.COSMIC_ID.unique():
```

```
    top10[cosmic_id]=sorted(res[cosmic_id], key=lambda x: x[1])[:10]
    truth[cosmic_id] = sorted(res[cosmic_id], key=lambda x: x[2])[:10]
    comparative_scores[cosmic_id] = score_rank_ordered_lst(truth[cosmic_id],␣
 ↪top10[cosmic_id])
# comparative_scores
```

[268]:
```
ten = pd.DataFrame()
for k, v in top10.items():
    lst = [el[0] for el in v]
    if len(lst) < 10:
        ten[k] = lst+[float("NaN")]*(10 - len(lst))
    else:
        ten[k]=lst

ten
```

[268]:
```
    924100  910924  687561  1287381  910922  905947  1287706  687452  \
0     1007    1007    1007      201     201     201     1007    1007
1      268     201    1004     1007     135    1494     1372     201
2      201    1494     201      180    1494     268     1004    1494
3     1494     140    1494     1004     140     135      201     268
4      180    1004    1003     1003     180     180     1494    1003
5      283     180    1248      140     197     140      140     140
6      140    1003    1031      197     200    1057     1003     180
7     1004     197     140     1031     194     197      180    1004
8     1003     200     268      136     283     200     1060     194
9     1031     194     283      194     268     194     1031    1031

    906798  906797  …  924248  909747  687586  687457  909907  \
0     1494     201  …     104     201     201    1007    1007
1     1007    1007  …     201    1007    1007     201    1031
2      201    1494  …     140     180    1494    1494    1004
3     1003    1003  …     180    1494     180    1004    1016
4      268     268  …     283    1372    1004     140    1057
5     1004     180  …     194     200    1003     180    1494
6      180    1004  …     197     197     140    1003    1003
7      140     140  …     200     140    1031     197    1026
8     1031     135  …     346     268    1057    1031    1166
9      194    1031  …      11    1003     197     268    1060

    909785  909904  909905  687592  946358
0      104    1007    1007    1007     104
1     1003     201    1004     201    1004
2     1004    1494    1494    1004    1003
3      140     140    1003     180    1494
4      283    1003    1031    1494    1248
5      201     268     283     140     201
```

```
6      1007       180       268      1031       283
7       268      1372      1016      1003      1007
8      1494      1004      1057      1248      1016
9      1248       194      1166       194       268

[10 rows x 962 columns]
```

0.7.2 Above are the top ten drugs predicted to be sensitive for each cell line. Each column header denotes a cell line, and the values of the column are the top ten drugs with the top being the drug predicted to be the most sensitive for that cell.

[274]:
```
comparative_scores
```

[274]:
```
{924100: 0.7,
 910924: 0.7,
 687561: 0.4,
 1287381: 0.6,
 910922: 0.3,
 905947: 0.6,
 1287706: 0.5,
 687452: 0.7,
 906798: 0.8,
 906797: 0.7,
 906800: 0.4,
 687562: 0.5,
 906795: 0.4,
 924102: 0.8,
 910921: 0.7,
 687563: 0.7,
 910935: 0.3,
 906793: 0.5,
 910784: 0.8,
 906792: 0.7,
 906794: 0.5,
 906804: 0.6,
 910697: 0.7,
 910934: 0.6,
 910851: 0.7,
 910925: 0.6,
 905948: 0.7,
 905949: 0.6,
 684052: 0.7,
 910920: 0.6,
 906791: 0.7,
 905950: 0.5,
 910944: 0.4,
 1295740: 0.8,
 910933: 0.3,
 1295741: 0.9,
 906790: 0.5,
 910687: 0.7,
 910704: 0.7,
 910781: 0.7,
 906765: 0.6,
 1290722: 0.5,
 910702: 0.7,
 753531: 0.5,
 753532: 0.5,
 753533: 0.9,
 910919: 0.3,
 910705: 0.7,
 906746: 0.5,
 924104: 0.8,
 906763: 0.5,
 753616: 0.5,
 910926: 0.8,
 910698: 0.9,
 906696: 0.3,
 753535: 0.6,
 1290724: 0.7,
 753534: 0.9,
 1240121: 0.4,
 1290725: 0.9,
 1240122: 0.5,
 910706: 0.5,
 906801: 0.6,
 946359: 0.5,
 949093: 0.4,
 905951: 0.9,
 910710: 0.4,
 924105: 0.6,
 906830: 0.6,
 910850: 0.8,
 906693: 0.7,
 687505: 0.7,
 687506: 0.8,
 910700: 0.4,
 906824: 0.6,
 910703: 0.6,
 753538: 0.8,
 753539: 0.6,
 905963: 0.6,
 1290730: 0.5,
 753541: 0.7,
 906826: 0.6,
 753540: 0.6,
 924106: 0.5,
 910916: 0.6,
 906831: 0.5,
 906832: 0.5,
 907295: 0.6,
 753551: 0.5,
 1297439: 0.7,
 946368: 0.7,
 906835: 0.6,
 1297446: 0.5,
 753545: 0.4,
 906834: 0.7,
 753546: 0.4,
 906836: 0.5,
 906842: 0.7,
 946369: 0.5,
 906837: 0.8,
 910936: 0.5,
 753547: 0.6,
 946370: 0.7,
 906838: 0.4,
 906843: 0.3,
 946372: 0.7,
 1479987: 0.6,
 906807: 0.9,
 946373: 0.7,
 753552: 0.6,
 1290795: 0.4,
 906841: 0.9,
 687448: 0.2,
 906805: 0.7,
 949088: 0.6,
 753549: 0.4,
 906839: 0.5,
 687780: 0.6,
 687983: 0.5,
 910937: 0.4,
 753548: 0.7,
 946377: 0.9,
 1297438: 0.7,
 910554: 0.1,
 1290797: 0.3,
 687980: 0.5,
 1322212: 0.5,
 906833: 0.8,
 687985: 0.6,
 1322213: 0.4,
 946367: 0.6,
 906808: 0.5,
 1303912: 0.5,
 1290769: 0.4,
 1290765: 0.8,
 910852: 0.5,
 1240123: 0.5,
 905952: 0.8,
 1240125: 0.6,
 1290771: 0.2,
 906814: 0.5,
 687777: 0.4,
 910688: 0.5,
 1240124: 0.7,
 910566: 0.5,
 906813: 0.5,
 724859: 0.5,
 906821: 0.7,
 910951: 0.8,
 905961: 0.6,
 906812: 0.5,
 946382: 0.6,
 910568: 0.5,
 687596: 0.5,
 910569: 0.4,
 949090: 0.1,
 910853: 0.7,
 910692: 0.5,
 924107: 0.7,
 753624: 0.7,
 910927: 0.7,
 910915: 0.7,
 910567: 0.7,
 910689: 0.5,
 924108: 0.7,
 910941: 0.5,
 906818: 0.8,
 910952: 0.3,
 906820: 0.5,
 906817: 0.5,
 906828: 0.9,
 910943: 0.5,
 1290767: 0.4,
 906827: 0.7,
 906823: 0.4,
 1290768: 0.7,
 910691: 0.5,
 907041: 0.3,
 907290: 0.5,
 1290814: 0.5,
 1240138: 0.3,
 906864: 0.5,
 1240129: 0.3,
 906869: 0.5,
 906999: 0.7,
 1290808: 0.5,
 1290806: 0.3,
 1240139: 0.7,
 1290807: 0.7,
 906870: 0.5,
 1290809: 0.6,
 1240140: 0.2,
 906871: 0.5,
 1290810: 0.1,
 1247873: 0.6,
 906865: 0.4,
 907299: 0.9,
 906872: 0.6,
 1240131: 0.8,
 907042: 0.4,
 907298: 0.7,
 906873: 0.6,
 1240132: 0.7,
 1240141: 0.3,
 949161: 0.6,
 949160: 0.7,
 1240130: 0.6,
 906875: 0.6,
 1290812: 0.4,
 1303896: 0.5,
 907291: 0.3,
 1240134: 0.5,
 906863: 0.4,
 910932: 0.5,
 906877: 0.6,
 1297449: 0.4,
 906868: 0.8,
 1290813: 0.6,
 687568: 0.5,
 1303897: 0.3,
 1240135: 0.7,
 1503361: 0.5,
 1240136: 0.6,
 1240137: 0.8,
 684057: 0.5,
 949162: 0.4,
 949166: 0.2,
 906846: 0.5,
 753554: 0.5,
 905970: 0.5,
 949157: 0.3,
 949165: 0.4,
 753555: 0.3,
 906855: 0.3,
 684059: 0.6,
 1503369: 0.7,
 949164: 0.5,
 906848: 0.8,
 1503370: 0.8,
 949155: 0.3,
 949167: 0.8,
 906849: 0.6,
 1240127: 0.7,
 1503366: 0.4,
 949168: 0.5,
 906851: 0.6,
 906856: 0.4,
 1503367: 0.5,
 1290798: 0.4,
 753556: 0.4,
 907000: 0.9,
 911905: 0.8,
 906861: 0.5,
 905935: 0.5,
 1240128: 0.6,
 906844: 0.5,
 906852: 0.5,
 949158: 0.5,
 906862: 0.5,
 906853: 0.3,
 949163: 0.4,
 906854: 0.6,
 684055: 0.8,
 906847: 0.5,
 1297447: 0.5,
 949156: 0.7,
 684062: 0.5,
 1298146: 0.4,
 753563: 0.7,
 1240155: 0.4,
 1322218: 0.7,
 905957: 0.5,
 907071: 0.5,
 907170: 0.6,
 1240149: 0.5,
 907064: 0.7,
 909977: 0.7,
 907289: 0.7,
 1240150: 0.5,
 907072: 0.6,
 1298141: 0.4,
 907065: 0.5,
 907285: 0.5,
 907066: 0.7,
 907286: 0.6,
 907067: 0.4,
 907171: 0.4,
 1298145: 0.6,
 907073: 0.6,
 753561: 0.5,
 1240151: 0.5,
 905939: 0.4,
 910779: 0.4,
 905972: 0.5,
 753562: 0.6,
 907068: 0.7,
 905973: 0.5,
 907287: 0.7,
 907169: 0.3,
 907060: 0.8,
 907061: 0.6,
 1240153: 0.7,
 1322224: 0.4,
 924151: 0.5,
 905968: 0.5,
 1298136: 0.7,
 907062: 0.4,
 907069: 0.6,
 1240154: 0.6,
 724869: 0.5,
 753559: 0.7,
 905936: 0.5,
 907056: 0.4,
 907044: 0.7,
 1240143: 0.6,
 1303900: 0.3,
 905938: 0.5,
 907046: 0.3,
 905937: 0.4,
 1240144: 0.9,
 749713: 0.8,
 907050: 0.6,
 907057: 0.6,
 1240145: 0.8,
 907047: 0.6,
 907058: 0.5,
 1290907: 0.7,
 749714: 0.3,
 924110: 0.6,
 907059: 0.7,
 1290922: 0.3,
 924111: 0.5,
 1290908: 0.5,
 749709: 0.3,
 1240146: 0.3,
 1290906: 0.4,
 907051: 0.8,
 907043: 0.6,
 749710: 0.3,
 749715: 0.8,
 907053: 0.6,
 1298134: 0.9,
 753558: 0.6,
 749711: 0.6,
 749716: 0.5,
 949153: 0.5,
 749712: 0.6,
 1240147: 0.7,
 905971: 0.7,
 1479988: 0.7,
 1240142: 0.7,
 907045: 0.3,
 749717: 0.6,
 907055: 0.9,
 1290905: 0.4,
 907048: 0.5,
 907312: 0.3,
 907311: 0.4,
 907320: 0.6,
 924187: 0.6,
 924188: 0.8,
 909975: 0.7,
 1298222: 0.6,
 909976: 0.7,
 1298215: 0.7,
 907321: 0.7,
 905989: 0.7,
 1330933: 0.5,
 907300: 0.4,
 753575: 0.7,
 753570: 0.7,
 1503368: 0.4,
 753576: 0.7,
 907280: 0.5,
 1298216: 0.7,
 1240166: 0.5,
 1330935: 0.6,
 1298218: 0.7,
 753573: 0.6,
 924239: 0.3,
 1298168: 0.6,
 1298219: 0.6,
 907317: 0.5,
 907277: 0.7,
 753618: 0.7,
 1298169: 0.4,
 753572: 0.8,
 907314: 0.6,
 907318: 0.4,
 907278: 0.7,
 1659817: 0.6,
 1240167: 0.5,
 907281: 0.7,
 907319: 0.8,
 924186: 0.4,
 907282: 0.7,
 907313: 0.5,
 753574: 0.6,
 907279: 0.6,
 753569: 0.9,
 907269: 0.4,
 907270: 0.4,
 907274: 0.6,
 907272: 0.6,
 753565: 0.4,
 1240161: 0.8,
 905940: 0.5,
 1240162: 0.3,
 1327775: 0.5,
 753566: 0.6,
 907275: 0.1,
 907175: 0.4,
 1298156: 0.9,
 907276: 0.4,
 907176: 0.9,
 1298157: 0.7,
 1298160: 0.6,
 1330931: 0.3,
 1327765: 0.9,
 1298151: 0.5,
 907268: 0.4,
 924238: 0.6,
 1240157: 0.7,
 1327766: 0.7,
 1240158: 0.6,
 1327768: 0.7,
 907271: 0.7,
 907172: 0.6,
 1240159: 0.7,
 1327773: 0.8,
 1240160: 0.5,
 1327769: 0.7,
 1327774: 0.6,
 907173: 0.8,
 998184: 0.7,
 907273: 0.6,
 753564: 0.6,
 1479995: 0.8,
 1327771: 0.4,
 1480358: 0.6,
 1480359: 0.9,
 908130: 0.7,
 1240174: 0.6,
 908139: 0.6,
 925340: 0.7,
 908123: 0.5,
 1240173: 0.6,
 905946: 0.5,
 925343: 0.6,
 924250: 0.9,
 1240178: 0.7,
 1330942: 0.4,
 910948: 0.5,
 908141: 0.6,
 925338: 0.2,
 687514: 0.2,
 908131: 0.5,
 908142: 0.4,
 908132: 0.9,
 905960: 0.7,
 1659818: 0.8,
 925339: 0.7,
 1330941: 0.1,
 908126: 0.4,
 908135: 0.7,
 905975: 0.5,
 908124: 0.8,
 908136: 0.5,
 683665: 0.4,
 908121: 0.7,
 908125: 0.4,
 924240: 0.5,
 908127: 0.7,
 724870: 0.6,
 1240172: 0.7,
 908128: 0.5,
 907799: 0.4,
 908122: 0.6,
 908129: 0.6,
 908138: 0.5,
 907790: 0.8,
 713878: 0.6,
 753589: 0.8,
 753577: 0.6,
 907787: 0.6,
 905974: 0.7,
 753578: 0.5,
 753586: 0.5,
 907791: 0.5,
 724863: 0.4,
 946361: 0.6,
 907796: 0.8,
 753579: 0.4,
 917486: 0.4,
 753592: 0.7,
 753581: 0.6,
 687787: 0.7,
 907792: 0.8,
 949094: 0.8,
 753582: 0.5,
 1240168: 0.4,
 998189: 0.6,
 753583: 0.5,
 1240169: 0.6,
 907794: 0.5,
 910694: 0.4,
 907322: 0.3,
 753584: 0.7,
 907795: 0.5,
 753585: 0.4,
 907788: 0.7,
 907323: 0.5,
 1240170: 0.7,
 753588: 0.6,
 907783: 0.2,
 1298223: 0.4,
 713899: 0.6,
 1298226: 0.6,
 949170: 0.3,
 907786: 0.5,
 907789: 0.6,
 1240182: 0.5,
 687807: 0.4,
 724873: 0.7,
 1240187: 0.3,
 908469: 0.6,
 908473: 0.4,
 1298349: 0.6,
 688011: 0.7,
 1240183: 0.6,
 910899: 1.0,
 687802: 0.4,
 722045: 0.8,
 688006: 0.6,
 688007: 0.4,
 688058: 0.6,
 753600: 0.5,
 1330972: 0.3,
 1330973: 0.4,
 1298348: 0.7,
 908474: 0.6,
 722058: 0.7,
 1240184: 0.7,
 687812: 0.5,
 908472: 0.6,
 908475: 0.5,
 1240185: 0.4,
 1330964: 0.9,
 724866: 0.7,
 908471: 0.6,
 687804: 0.3,
 687798: 0.6,
 1240186: 0.3,
 1298350: 0.4,
 688010: 0.6,
 684681: 0.3,
 687799: 0.4,
 724868: 0.6,
 924244: 0.6,
 688001: 0.4,
 687800: 0.8,
 908476: 0.4,
 910900: 0.4,
 908463: 0.4,
 1298347: 0.7,
 949172: 0.5,
 949177: 0.8,
 687995: 0.6,
 687997: 0.4,
 1330947: 0.3,
 1330948: 0.8,
 908156: 0.4,
 949178: 0.7,
 1330950: 0.4,
 753595: 0.5,
 908468: 0.9,
 908467: 0.7,
 908146: 0.8,
 949175: 0.8,
 908147: 0.9,
 971777: 0.6,
 1323913: 0.4,
 724831: 0.4,
 753599: 0.4,
 905958: 0.7,
 908148: 0.6,
 753596: 0.6,
 949176: 0.8,
 908158: 0.5,
 949173: 0.6,
 908150: 0.7,
 908143: 0.6,
 908159: 0.6,
 908440: 0.5,
 908151: 0.4,
 1240181: 0.4,
 949174: 0.3,
 908149: 0.8,
 753594: 0.3,
 949179: 0.3,
 908144: 0.6,
 1240179: 0.9,
 1509073: 0.8,
 908145: 0.7,
 908152: 0.4,
 949171: 0.6,
 1509074: 0.7,
 908455: 0.7,
 1298357: 0.5,
 1330985: 0.6,
 688031: 0.4,
 908447: 0.6,
 910849: 0.5,
 909256: 0.6,
 753607: 0.7,
 908448: 0.4,
 1503363: 0.3,
 910399: 0.7,
 1298358: 0.5,
 1240192: 0.3,
 908449: 0.8,
 1503362: 0.5,
 910079: 0.7,
 1240193: 0.5,
 908452: 0.3,
 946360: 0.8,
 1298359: 0.6,
 910548: 0.7,
 910549: 0.8,
 724825: 0.6,
 908450: 0.5,
 909257: 0.7,
 908461: 0.6,
 908451: 0.7,
 910947: 0.4,
 687600: 0.5,
 908444: 0.5,
 908446: 0.4,
 925345: 0.3,
 1290455: 0.6,
 724855: 0.6,
 908454: 0.6,
 908457: 0.7,
 908445: 0.4,
 1330983: 0.9,
 910942: 0.8,
 1330982: 0.6,
 1330984: 0.6,
 909194: 0.6,
 925341: 0.6,
 1659819: 0.5,
 908465: 0.6,
 688026: 0.5,
 724874: 0.3,
 905942: 0.5,
 688022: 1.0,
 908460: 0.7,
 1240191: 0.7,
 1298352: 0.5,
 688023: 0.7,
 722066: 0.9,
 687819: 0.5,
 687829: 0.8,
 688014: 0.6,
 687820: 0.7,
 688027: 0.8,
 687815: 0.7,
 687821: 0.5,
 753605: 0.5,
 908458: 0.2,
 1298353: 0.3,
 1298356: 0.6,
 908443: 0.5,
 908462: 0.5,
 905944: 0.5,
 724834: 0.6,
 908481: 0.5,
 924241: 1.0,
 908470: 0.5,
 908483: 0.4,
 688013: 0.5,
 688018: 0.5,
 753604: 0.5,
 688025: 0.7,
 1240189: 0.5,
 687816: 0.5,
 1240190: 0.6,
 905941: 0.4,
 905967: 0.3,
 908459: 0.4,
 1298351: 0.6,
 908482: 0.6,
 722046: 0.8,
 688021: 0.5,
 1298539: 0.4,
 909698: 0.7,
 909702: 0.6,
 910544: 0.4,
 910545: 0.5,
 909262: 0.6,
 909263: 0.6,
 909264: 0.7,
 910861: 0.5,
 909703: 0.5,
 910401: 0.6,
 909696: 0.7,
 909704: 0.6,
 930082: 0.3,
 687455: 0.5,
 1330995: 0.4,
 1240209: 0.8,
 909699: 0.5,
 1524418: 0.4,
 1298537: 0.6,
 930083: 0.8,
 909706: 0.4,
 1524417: 0.7,
 910931: 0.4,
 1331025: 0.4,
 905978: 0.6,
 1524414: 0.6,
 1298538: 0.4,
 1330991: 0.4,
 1240208: 0.8,
 1524415: 0.8,
 909700: 0.7,
 1298533: 0.6,
 909697: 0.2,
 909701: 0.3,
 1524416: 0.8,
 971773: 0.8,
 1524419: 0.7,
 910903: 0.5,
 910546: 0.6,
 909974: 0.5,
 971774: 0.6,
 905964: 0.3,
 1330993: 0.7,
 1330994: 0.8,
 1240210: 0.5,
 1298534: 0.3,
 1298526: 0.3,
 1240201: 0.8,
 1298531: 0.6,
 1240206: 0.6,
 1298362: 0.5,
 1240196: 0.6,
 1240198: 0.4,
 1298475: 0.7,
 1480371: 0.6,
 925346: 0.6,
 1240207: 0.3,
 1480361: 0.6,
 1240199: 0.3,
 1298476: 0.5,
 1480372: 0.7,
 1240200: 0.8,
 909260: 0.6,
 1480362: 0.9,
 1298365: 0.6,
 925347: 0.4,
 1480360: 1.0,
 925348: 1.0,
 683667: 0.9,
 1240197: 0.5,
 909251: 0.8,
 753608: 0.8,
 1480364: 0.3,
 909252: 0.4,
 905934: 0.7,
 909248: 0.7,
 909253: 0.5,
 1240202: 0.8,
 905933: 0.8,
 1330987: 0.5,
 1240204: 0.6,
 909249: 0.5,
 905990: 0.9,
 1298529: 0.6,
 909250: 0.5,
 905969: 0.6,
 909255: 0.8,
 1240205: 0.7,
 905991: 0.6,
 909740: 0.5,
 1674021: 0.8,
 1240218: 0.7,
 909735: 0.8,
 1240219: 0.4,
 905959: 0.4,
 1659823: 0.6,
 1659928: 0.7,
 905965: 0.8,
 1247871: 0.6,
 909731: 0.7,
 1240216: 0.6,
 910906: 0.2,
 909736: 0.6,
 910209: 0.8,
 1331039: 0.8,
 909732: 0.6,
 1299061: 0.7,
 930298: 0.6,
 1240217: 0.7,
 909742: 0.4,
 1331040: 0.7,
 1660034: 0.6,
 1331033: 0.7,
 909743: 0.3,
 909722: 0.4,
 1240215: 0.5,
 909737: 0.2,
 1331034: 0.5,
 909729: 0.8,
 753612: 0.6,
 909738: 0.5,
 1331035: 0.2,
 1299059: 0.5,
 909739: 0.5,
 724828: 0.6,
 1331032: 0.5,
 1660035: 0.1,
 1331036: 0.4,
 688086: 0.8,
 1660036: 0.6,
 1331037: 0.7,
 688087: 0.6,
 905979: 0.8,
 910905: 0.1,
 905982: 0.7,
 1331038: 0.5,
 717431: 0.6,
 1240212: 0.8,
 909716: 0.5,
 909725: 0.3,
 905954: 0.4,
 713885: 0.6,
 909717: 0.6,
 909724: 0.3,
 753610: 0.6,
 909712: 0.5,
 713880: 0.6,
 909718: 0.3,
 909726: 0.6,
 905986: 0.6,
 684072: 0.6,
 909727: 0.8,
 905956: 0.8,
 910911: 0.5,
 905985: 0.5,
 1503364: 0.5,
 1503365: 0.7,
 909728: 0.4,
 910701: 0.5,
 905984: 0.5,
 909719: 0.7,
 910930: 0.6,
 909713: 0.5,
 909720: 0.8,
 910904: 0.7,
 724872: 0.7,
 909721: 0.6,
 909707: 0.8,
 909709: 0.3,
 1299052: 0.6,
 909715: 0.5,
 909723: 0.4,
 909711: 0.2,
 930297: 0.5,
 905955: 0.6,
 909708: 0.6,
 753620: 0.7,
 1299050: 0.7,
 909779: 0.4,
 1480374: 0.7,
 1299078: 0.3,
 905980: 0.5,
 753615: 0.6,
 909777: 0.6,
 688121: 0.7,
 1240226: 0.8,
 687590: 0.5,
 1331048: 0.8,
 1299080: 0.6,
 1299070: 0.7,
 905981: 0.6,
 1299075: 0.7,
 1299081: 0.6,
 905983: 0.7,
 1240222: 0.5,
 909780: 0.7,
 1331050: 0.5,
 930299: 0.4,
 905977: 0.6,
 713869: 0.9,
 1240223: 0.8,
 905976: 0.4,
 909769: 0.5,
 910910: 0.7,
 910695: 0.8,
 909773: 0.7,
 909778: 0.6,
 909781: 0.4,
 910780: 0.4,
 909774: 0.6,
 1331049: 0.5,
 909784: 0.5,
 909771: 0.4,
 687588: 0.3,
 724838: 0.6,
 909776: 0.6,
 1331045: 0.6,
 735784: 0.9,
 946355: 0.5,
 909748: 0.4,
 1240220: 0.8,
 909755: 0.7,
 909756: 0.5,
 1240221: 0.6,
 753623: 0.4,
 724878: 0.6,
 724879: 0.8,
 909749: 0.6,
 909757: 0.6,
 946357: 0.6,
 687459: 0.7,
 946353: 0.5,
 753621: 0.7,
 909750: 0.6,
 924247: 0.6,
 910907: 0.6,
 909758: 0.5,
 753622: 0.7,
 909770: 0.7,
 946354: 0.6,
 909751: 0.7,
 909759: 0.8,
 905962: 0.8,
 724812: 0.7,
 946356: 0.3,
 909745: 0.5,
 909753: 0.6,
 1299064: 0.8,
 909746: 0.5,
 753614: 0.3,
 1299062: 0.6,
 909754: 0.5,
 905945: 0.5,
 1503371: 0.6,
 909744: 0.8,
 724839: 0.6,
 909761: 0.4,
 924248: 0.3,
 909747: 0.5,
 687586: 0.7,
 687457: 0.6,
 909907: 0.6,
 909785: 0.4,
 909904: 0.7,
 909905: 0.7,
 687592: 0.4,
 946358: 0.2}
```

0.7.3 Above are the scores for comparing my rank ordered list to the rank ordering of IC50 from the Drug Response Data. Closer to 1 means higher similarity.

0.8 Part 3 summary:

0.8.1 Although some drug models perform worse than others, the sensitivity rank ordered top 10 lists were fairly comparable to the ground truth in the Drug Response Data. With more time, the models and data could be improved; however, surprisingly, the SIR classifications performed well (~.7 in accuracy) and perhaps even better than the regression (although not directly comparable).

```
[ ]:
```