

A Microbial-Only Random Forest Classifier for CRC Anatomical Side

August 24, 2022

```
[1]: #Created by: Tyler Kolisnik
      #March 27, 2022

      # Description:
      #This code is for evaluating and finalizing a random forest model after using
      ↪grid search cv to find the best parameters
      #Includes scoring and validation

      # Import required packages and functions
      import sklearn
      import numpy as np
      import pandas as pd
      import rpy2.robjects as robjects
      import pickle
      import seaborn as sns
      import matplotlib.pyplot as plt
      from sklearn.ensemble import RandomForestClassifier
      from sklearn.model_selection import train_test_split
      from sklearn import metrics
      from rpy2.robjects import pandas2ri
      from sklearn.metrics import confusion_matrix, make_scorer
      from sklearn.metrics import classification_report
      from sklearn.metrics import roc_curve
      from matplotlib import pyplot

      # Increase output plot resolution
      plt.rcParams['figure.dpi'] = 300
      plt.rcParams['savefig.dpi'] = 300
      sns.set(rc={"figure.dpi":300, 'savefig.dpi':300})
      sns.set_context('notebook')
      sns.set_style("whitegrid")

      # Import the training set data (278 samples)
      pandas2ri.activate()
      readRDS = robjects.r['readRDS']
```

```

ml_data=readRDS('/Volumes/FryShareNVME/2022-03-25-ML-Results-Side/
↳2022-03-27-finalized-paper-data/microbial-rf-exploration /
↳2022-05-29-microbial-only preprocessing workflow_SIDE/output_data/
↳2022-03-29-CPM-MicrobialONLY-discovery-set-side-278samples.rds')
tpmdata=ml_data[0]
tpmdata2=tpmdata.transpose()
targetdata=np.ravel(ml_data[1]).astype(int)
targetlabels=np.ravel(ml_data[2])

```

```

[2]: # Create the classifier from the best parameter set found in the grid search CV
rforest_finalparamset = RandomForestClassifier(
    bootstrap=True,
    class_weight=None,
    criterion='gini',
    max_depth=5,
    max_features=0.1,
    max_leaf_nodes=10,
    min_impurity_decrease=0.0,
    min_samples_leaf=1,
    min_samples_split=2,
    min_weight_fraction_leaf=0.0,
    n_estimators=100,
    n_jobs=2,
    oob_score=True,
    random_state=2,
    verbose=0,
    warm_start=False
)

```

```

[3]: # Split the dataset into training and testing
#Test Size = 0.25 instructs the model to fit on 75% of the data and test on 25%
#Stratify ensures an even distribution of samples
#Fixing random state allows for reproducibility of results
X_train, X_test, y_train, y_test = train_test_split(tpmdata2, targetdata,
                                                    test_size=0.
                                                    ↳25,random_state=1,
                                                    ↳stratify=targetdata,shuffle=True)

# Fit (train) the classifier to our dataset
#alternatively import this from a file if you have already saved the model (.
↳pkl)
rforest_finalparamset.fit(X_train,y_train)

# Load the model from disk
#filename = '/Volumes/FryShareNVME/2022-03-25-ML-Results-Side/
↳2022-03-26-ML-paramset-analysis/crc-side-random-forest-model.pkl'

```

```

#rforest_finalparamset = pickle.load(open(filename, 'rb'))
#rforest_finalparamset2.score(X_test, Y_test)
# Save the model to disk
#filename = '/Volumes/FryShareNVME/2022-03-25-ML-Results-Side/
↳2022-03-26-ML-paramset-analysis/crc-side-random-forest-model.pkl'
#pickle.dump(rforest_finalparamset, open(filename, 'wb'))

# Make predictions from the training data
rfpredictions=rforest_finalparamset.predict(X_test)
print("Model Accuracy:",metrics.accuracy_score(y_test,rfpredictions))

```

Model Accuracy: 0.7571428571428571

```

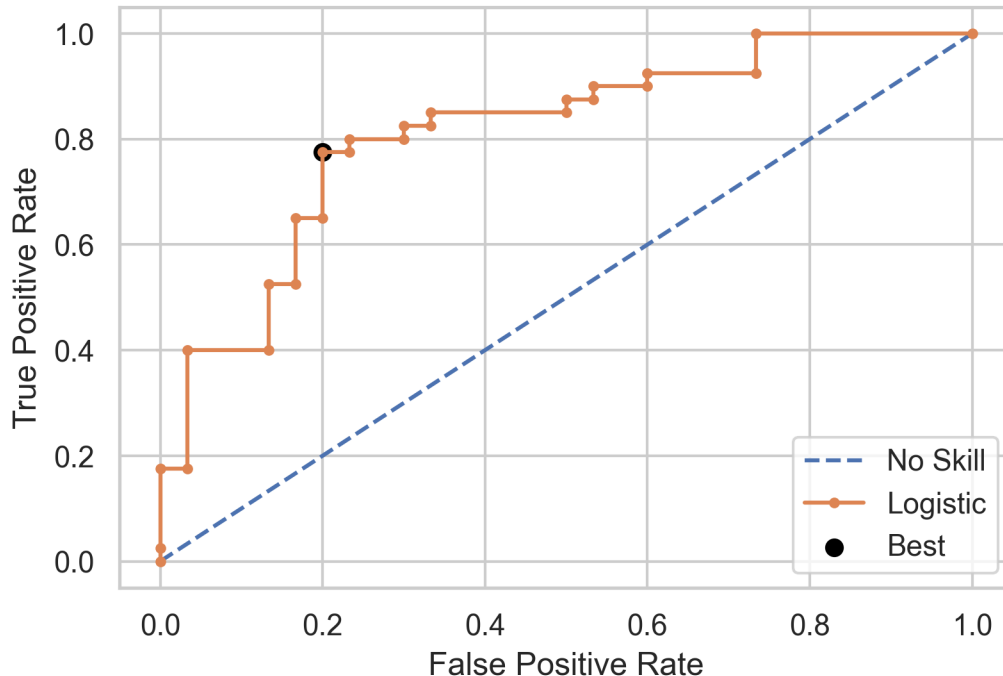
[4]: # Threshold hyperparameter optimization
#The default threshold for a binary variable is 0.5
#Checking the threshold allows for the optimization of the sensitivity and
↳specificity
#and a correction for imbalanced datasets
#If a new threshold is set then you must no longer use .predict, and must use .
↳predict_proba

# Generate prediction probabilities (necessary for AUROC/AUPRC and thresholds)
yhat = rforest_finalparamset.predict_proba(X_test)
# keep probabilities for the positive outcome only
yhat = yhat[:, 1]
# calculate roc curves
fpr, tpr, thresholds = roc_curve(y_test, yhat)
# calculate the g-mean for each threshold
gmeans = np.sqrt(tpr * (1-fpr))
# locate the index of the largest g-mean
ix = np.argmax(gmeans)
print('Best Threshold=%f, G-Mean=%.3f' % (thresholds[ix], gmeans[ix]))
# plot the roc curve for the model
pyplot.plot([0,1], [0,1], linestyle='--', label='No Skill')
pyplot.plot(fpr, tpr, marker='.', label='Logistic')
pyplot.scatter(fpr[ix], tpr[ix], marker='o', color='black', label='Best')
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
pyplot.legend()
# show the plot
pyplot.show()

```

#With Code Adapted From: <https://machinelearningmastery.com/threshold-moving-for-imbalanced-classification/>

Best Threshold=0.530549, G-Mean=0.787



```
[5]: # See how a threshold adjustment affects the training set
#I ultimately chose to just keep the 0.5 threshold as the scores are close
#enough
#and it is not of utmost importance in this model to prioritize
#precision(specificity) or recall(sensitivity) over the other

threshold = 0.530549
thresholdpredictions = rforest_finalparamset.predict_proba(X_test)
discpreds2 = thresholdpredictions[:, 1]
test_predictions=rforest_finalparamset.predict(X_test)

preds = [1 if thresholdpredictions[i][1]> threshold else 0 for i in
#range(len(thresholdpredictions))]

print("0.530549 Threshold:")
pd.DataFrame(classification_report(y_test,preds,output_dict=True)).T
```

0.530549 Threshold:

```
[5]:
```

	precision	recall	f1-score	support
0	0.705882	0.800000	0.750000	30.000000
1	0.833333	0.750000	0.789474	40.000000
accuracy	0.771429	0.771429	0.771429	0.771429
macro avg	0.769608	0.775000	0.769737	70.000000
weighted avg	0.778711	0.771429	0.772556	70.000000

```
[6]: print("0.5 Threshold:")
pd.DataFrame(classification_report(y_test,test_predictions,output_dict=True)).T
```

0.5 Threshold:

```
[6]:
```

	precision	recall	f1-score	support
0	0.724138	0.700000	0.711864	30.000000
1	0.780488	0.800000	0.790123	40.000000
accuracy	0.757143	0.757143	0.757143	0.757143
macro avg	0.752313	0.750000	0.750994	70.000000
weighted avg	0.756338	0.757143	0.756584	70.000000

```
[7]: # Print Scoring Metrics for Training (Model Discovery) Set
print("Accuracy:",metrics.accuracy_score(y_test,rfpredictions))
print("Out-of-Bag Score:",rforest_finalparamset.oob_score_)
print("F1 Score:",metrics.f1_score(y_test,rfpredictions))
print("ROC AUC Score:",metrics.roc_auc_score(y_test,rfpredictions))
print("Recall Score:",metrics.recall_score(y_test,rfpredictions))
print("Precision Score:",metrics.precision_score(y_test,rfpredictions))
confmat = confusion_matrix(y_test,rfpredictions)
fp = confmat[0][1]
tn = confmat[0][0]
tp = confmat[1][1]
fn = confmat[1][0]
tprate=tp/(tp+fn)
fprate=fp/(fp+tn)

print("False Positives:",fp)
print("False Negatives:",fn)
print("True Positives:",tp)
print("True Negatives:",tn)
print("True Positive Rate:",tprate)
print("True Negative Rate:",fprate)

print(classification_report(y_test,rfpredictions))
```

Accuracy: 0.7571428571428571
Out-of-Bag Score: 0.7355769230769231
F1 Score: 0.7901234567901235
ROC AUC Score: 0.75
Recall Score: 0.8

Precision Score: 0.7804878048780488

False Positives: 9

False Negatives: 8

True Positives: 32

True Negatives: 21

True Positive Rate: 0.8

True Negative Rate: 0.3

	precision	recall	f1-score	support
0	0.72	0.70	0.71	30
1	0.78	0.80	0.79	40
accuracy			0.76	70
macro avg	0.75	0.75	0.75	70
weighted avg	0.76	0.76	0.76	70

```
[8]: # Generate AUROC/AUPRC scores and create functions for plotting curves
valauprc = sklearn.metrics.
    ↳average_precision_score(y_test,rfpredictions,pos_label=1)

valauroc = sklearn.metrics.roc_auc_score(y_test,rfpredictions)
aurocscore="AUROC="+str(round(valauroc,2))
auprcscore="AUPRC="+str(round(valauprc,2))
print(auprcscore)
print(aurocscore)
def plot_auroc_curve(y_test,rfpredictions,aurocscore):
    data_fpr, data_tpr, data_thresholds = sklearn.metrics.
    ↳roc_curve(y_test,rfpredictions)
    data_fpr_tpr = pd.DataFrame({'fpr':data_fpr, 'tpr':data_tpr})
    p = sns.lineplot(data=data_fpr_tpr, x='fpr', y='tpr',ci=None)
    p.set(xlabel='False Positive Rate', ylabel='True Positive Rate')
    p.plot([0, 1], [0, 1], color='black', ls='--')
    p.text(0.2,0.5,aurocscore)

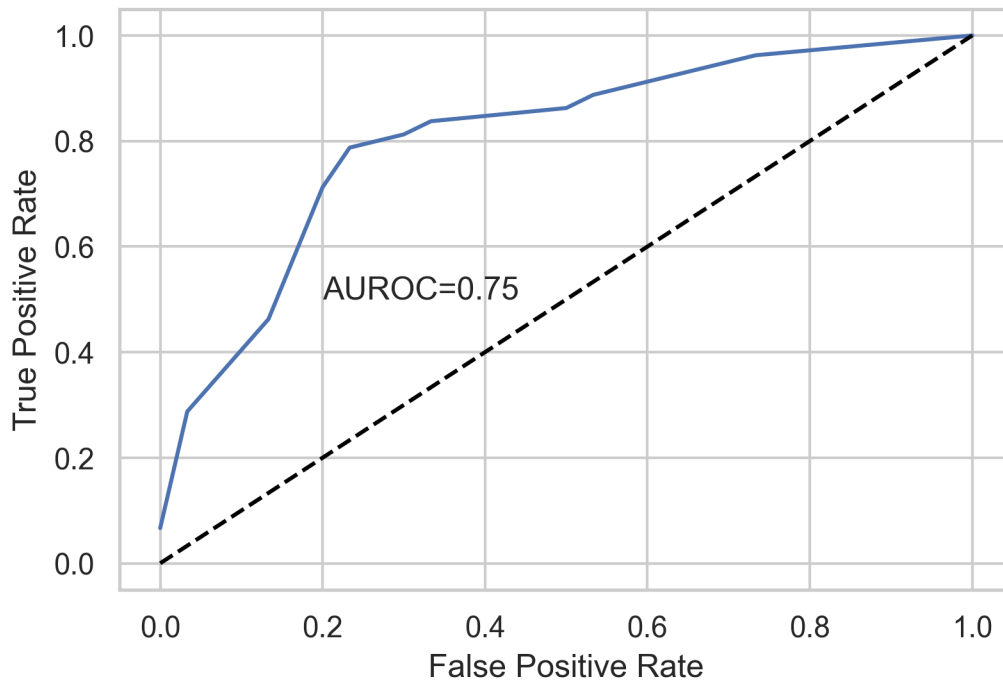
def plot_auprc_curve(y_test,rfpredictions,auprcscore):
    data_prcsn, data_rcll, data_thrshlds = sklearn.metrics.
    ↳precision_recall_curve(y_test,rfpredictions)
    data_prcsn_rcll = pd.DataFrame({'prcsn':data_prcsn, 'rcll':data_rcll})
    p = sns.lineplot(data=data_prcsn_rcll, x='rcll', y='prcsn',ci=None)
    p.set(xlabel='Recall', ylabel='Precision')
    p.plot([0, 1], [1, 0], color='black', ls='--')
    p.text(0.2,0.5,auprcscore)
```

AUPRC=0.74

AUROC=0.75

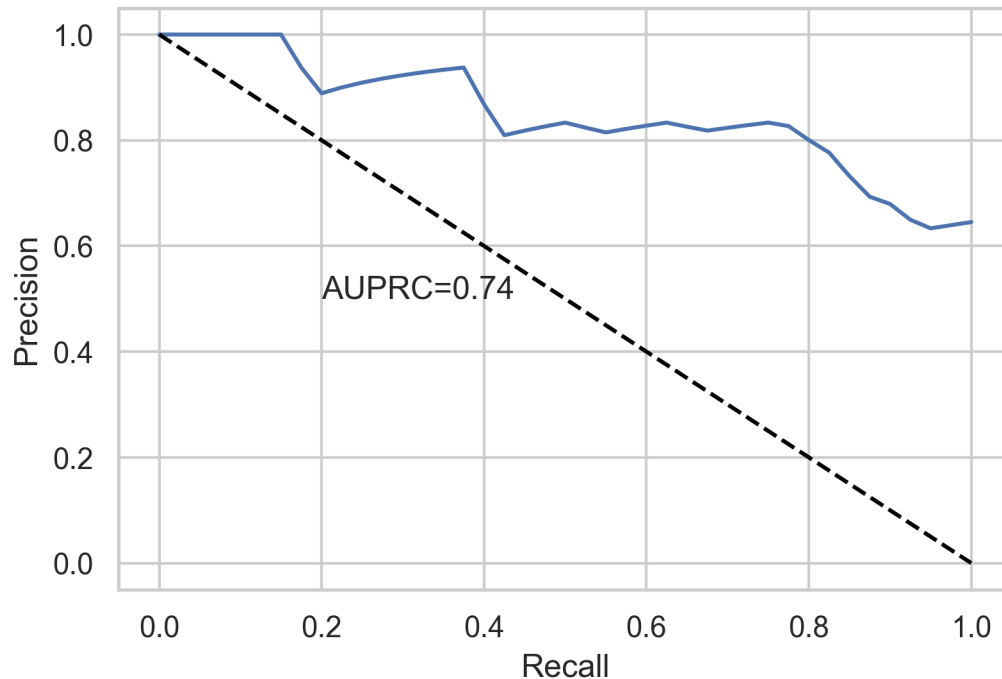
```
[10]: # Plot AUROC Curve
plot_auroc_curve(y_test, discpreds2, aurocscore)

# Save to file
plt.savefig("/Volumes/FryShareNVME/2022-03-25-ML-Results-Side/
↳2022-03-27-finalized-paper-data/microbial-rf-exploration /
↳microbialonly-AUROC-training-set.pdf")
```



```
[11]: # Plot AUPRC Curve
plot_auprc_curve(y_test, discpreds2, auprcscore)

# Save to file
plt.savefig("/Volumes/FryShareNVME/2022-03-25-ML-Results-Side/
↳2022-03-27-finalized-paper-data/microbial-rf-exploration /
↳microbialonly-AUPRC-training-set.pdf")
```



```
[12]: # Import independent validation set (30 samples the model has never seen before)
pandas2ri.activate()
readRDS = robjects.r['readRDS']
val_data=readRDS('/Volumes/FryShareNVME/2022-03-25-ML-Results-Side/
↳2022-03-27-finalized-paper-data/microbial-rf-exploration /
↳2022-05-29-microbial-only preprocessing workflow_SIDE/output_data/
↳2022-03-29-CPM-MicrobialONLY-validation-set-side-30samples.rds')
valdata=val_data[0]
valdata2=valdata.transpose()
valtargetdata=np.ravel(val_data[1]).astype(int)

[13]: # Run the model on the validation set
runmodelonvaldata=rforest_finalparamset.predict(valdata2)

[14]: # Generate prediction probabilities (necessary for AUROC/AUPRC and thresholds)
val_prob_preds = rforest_finalparamset.predict_proba(valdata2)

# Take only the positive values
val_preds = val_prob_preds[:, 1]

[15]: # Print Scoring Metrics for Validation Set
print("Accuracy:",metrics.accuracy_score(runmodelonvaldata, valtargetdata))
print("F1 Score:",metrics.f1_score(runmodelonvaldata, valtargetdata))
print("ROC AUC Score:",metrics.roc_auc_score(runmodelonvaldata, valtargetdata))
```



```

print("Recall Score:",metrics.recall_score(runmodelonvaldata, valtargetdata))
print("Precision Score:",metrics.precision_score(runmodelonvaldata,
↪valtargetdata))
confmat = confusion_matrix(runmodelonvaldata,valtargetdata)
fp = confmat[0][1]
tn = confmat[0][0]
tp = confmat[1][1]
fn = confmat[1][0]
tprate=tp/(tp+fn)
fprate=fp/(fp+tn)

print("False Positives:",fp)
print("False Negatives:",fn)
print("True Positives:",tp)
print("True Negatives:",tn)
print("True Positive Rate:",tprate)
print("True Negative Rate:",fprate)

print(classification_report(runmodelonvaldata, valtargetdata))

```

```

Accuracy: 0.7
F1 Score: 0.7567567567567568
ROC AUC Score: 0.7556818181818181
Recall Score: 0.6363636363636364
Precision Score: 0.9333333333333333
False Positives: 1
False Negatives: 8
True Positives: 14
True Negatives: 7
True Positive Rate: 0.6363636363636364
True Negative Rate: 0.125

```

	precision	recall	f1-score	support
0	0.47	0.88	0.61	8
1	0.93	0.64	0.76	22
accuracy			0.70	30
macro avg	0.70	0.76	0.68	30
weighted avg	0.81	0.70	0.72	30

```

[16]: # Generate validation set AUROC/AUPRC scores
val_auprc = sklearn.metrics.average_precision_score(runmodelonvaldata,
↪valtargetdata,pos_label=1)

val_auroc = sklearn.metrics.roc_auc_score(runmodelonvaldata, valtargetdata)
aurocscore="AUROC="+str(round(val_auroc,2))

```

```

auprcscore="AUPRC="+str(round(val_auprc,2))
print(auprcscore)
print(aurocscore)

```

AUPRC=0.86

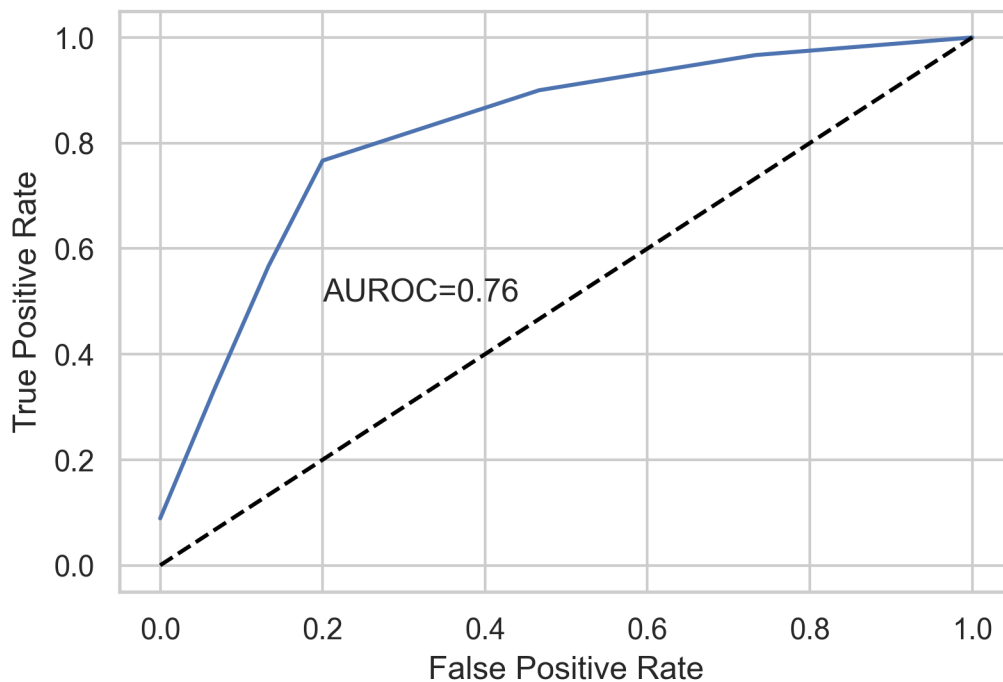
AUROC=0.76

```

[17]: # Plot AUROC Curve
auroccurve=plot_auroc_curve(valtargetdata,val_preds,aurocscore)

# Save to file
plt.savefig("/Volumes/FryShareNVME/2022-03-25-ML-Results-Side/
↳2022-03-27-finalized-paper-data/microbial-rf-exploration /
↳microbialonly-AUROC-validation-set.pdf")

```

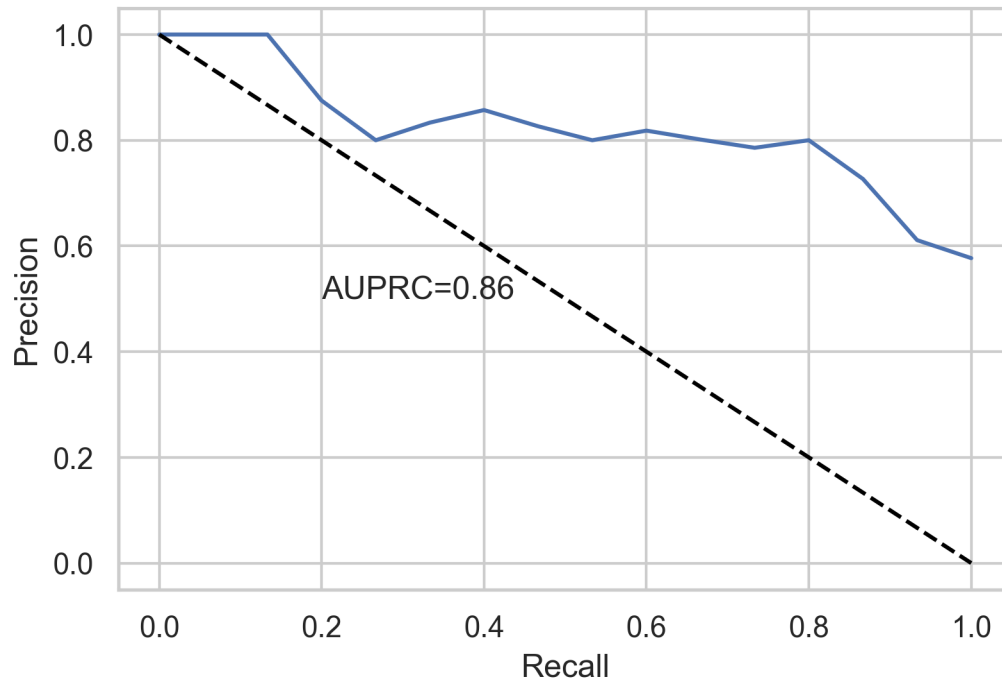


```

[18]: # Plot AUPRC Curve
plot_auprc_curve(valtargetdata,val_preds,auprcscore)

# Save to file
plt.savefig("/Volumes/FryShareNVME/2022-03-25-ML-Results-Side/
↳2022-03-27-finalized-paper-data/microbial-rf-exploration /
↳microbialonly-AUPRC-validation-set.pdf")

```



```
[19]: # See how a threshold adjustment affects the validation set (as a post-measure
      ↪ of performance, not for threshold optimization)
      #I ultimately chose to just keep the 0.5 threshold as the scores are close
      ↪ enough
      #and it is not of utmost importance in this model to prioritize
      #precision(specificity) or recall(sensitivity) over the other

      threshold = 0.539077 # define threshold here
      thresholdpredictions = rforest_finalparamset.predict_proba(valdata2)
      valpredictions=rforest_finalparamset.predict(valdata2)

      preds = [1 if thresholdpredictions[i][1]> threshold else 0 for i in
      ↪ range(len(thresholdpredictions))]

      # Print Classification Report for New Threshold
      print("0.530549 Threshold:")
      pd.DataFrame(classification_report(valtargetdata,preds,output_dict=True)).T
```

0.530549 Threshold:

```
[19]:
```

	precision	recall	f1-score	support
0	0.833333	0.666667	0.740741	15.000000
1	0.722222	0.866667	0.787879	15.000000
accuracy	0.766667	0.766667	0.766667	0.766667

macro avg	0.777778	0.766667	0.764310	30.000000
weighted avg	0.777778	0.766667	0.764310	30.000000

```
[20]: # Print Classification Report for 0.5 Threshold (Default)
print("0.5 Threshold:")
pd.
↳ DataFrame(classification_report(valtargetdata, valpredictions, output_dict=True)).
↳ T
```

0.5 Threshold:

```
[20]:
```

	precision	recall	f1-score	support
0	0.875000	0.466667	0.608696	15.0
1	0.636364	0.933333	0.756757	15.0
accuracy	0.700000	0.700000	0.700000	0.7
macro avg	0.755682	0.700000	0.682726	30.0
weighted avg	0.755682	0.700000	0.682726	30.0

```
[21]: # Save the feature importance scores of the random forest model mapped to the
↳ features (genes)
ftnames=rforest_finalparamset.feature_names_in_
ftimportances=rforest_finalparamset.feature_importances_
df=pd.DataFrame({ 'feature_names':ftnames, 'feature_importances':ftimportances,})
#df.shape
# Save to file
df.to_csv("/Volumes/FryShareNVME/2022-03-25-ML-Results-Side/
↳ 2022-03-27-finalized-paper-data/microbial-rf-exploration /
↳ 2022-04-05-microbialonly-crc-rf-final-feature-importance-scores.csv")
```