classic-pred

June 6, 2025

[13]:
```
import pandas as pd
import numpy as np
import os
import warnings
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
from xgboost import XGBClassifier
from tqdm import tqdm
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import (
    f1_score, accuracy_score, recall_score, precision_score,
    precision_recall_curve, confusion_matrix, roc_auc_score,
    matthews_corrcoef, roc_curve
)

warnings.filterwarnings("ignore")
```

0.0.1 Metric Calculation Utilities

[14]:
```
def get_aupr(pre, rec):
    pr_value = 0.0
    for ii in range(len(rec[:-1])):
        x_r, x_l = rec[ii], rec[ii+1]
        y_t, y_b = pre[ii], pre[ii+1]
        tempo = abs(x_r - x_l) * (y_t + y_b) * 0.5
        pr_value += tempo
    return pr_value

def scores(y_test, y_pred, th=0.5):
    y_predlabel = [(0. if item < th else 1.) for item in y_pred]
    tn, fp, fn, tp = confusion_matrix(y_test, y_predlabel).flatten()
    SPE = tn / (tn + fp)
    MCC = matthews_corrcoef(y_test, y_predlabel)
    fpr, tpr, _ = roc_curve(y_test, y_pred)
```

```
    sen, spe, pre, f1, mcc, acc, auc, tn, fp, fn, tp = np.array([
        recall_score(y_test, y_predlabel), SPE, precision_score(y_test,␣
↪y_predlabel),
        f1_score(y_test, y_predlabel), MCC, accuracy_score(y_test, y_predlabel),
        roc_auc_score(y_test, y_pred), tn, fp, fn, tp
    ])
    precision, recall, _ = precision_recall_curve(y_test, y_pred)
    aupr = get_aupr(precision, recall)
    return [aupr, auc, f1, acc, sen, spe, pre, fpr, tpr, precision, recall]
```

1 — Utility Functions —

[15]:
```
def combine_features(phage_dna, host_dna, phage_pro, host_pro):
    combined = np.concatenate([phage_dna, host_dna, phage_pro, host_pro],␣
↪axis=1)
    return combined
```

[16]:
```
def load_feature_vector(file_path):
    return np.loadtxt(file_path)
```

[17]:
```
def obtain_features(phage_list, host_list, labels, dna_base, pro_base):
    X_phage_dna, X_host_dna = [], []
    X_phage_pro, X_host_pro = [], []
    for p, h in zip(phage_list, host_list):
        X_phage_dna.append(load_feature_vector(os.path.join(dna_base, 'phage',␣
↪f'{p}.txt')))
        X_host_dna.append(load_feature_vector(os.path.join(dna_base,␣
↪'bacteria', f'{h}.txt')))
        X_phage_pro.append(load_feature_vector(os.path.join(pro_base, 'phage',␣
↪f'{p}.txt')))
        X_host_pro.append(load_feature_vector(os.path.join(pro_base,␣
↪'bacteria', f'{h}.txt')))
    return (np.array(X_phage_dna), np.array(X_host_dna),
            np.array(X_phage_pro), np.array(X_host_pro),
            np.array(labels))
```

2 — Load interaction matrix —

[6]:
```
interaction_matrix_path = "../ordinal_dataset_features/interaction_matrix.xlsx"
csv_output_path = "../ordinal_dataset_features/interaction_matrix.csv"

dna_base = '../ordinal_dataset_features/dna_features_ordinal_data'
pro_base = '../ordinal_dataset_features/prot_features_ordinal_data'
```

[18]:
```
interaction_matrix_path = "../phl_dataset_features/phage_host_interactions (1).
 ↪csv"
csv_output_path = "../phl_dataset_features/phage_host_interactions (1).csv"

dna_base = '../phl_dataset_features/dna_features'
pro_base = '../phl_dataset_features/protein_features'
```

[19]:
```
# Get list of phages and hosts based on files present
if interaction_matrix_path.endswith('.xlsx'):
    #Transposed because this dataset has another the wrong shape
    df = pd.read_excel(interaction_matrix_path, index_col=0).T
    df.to_csv(csv_output_path)
    interaction_matrix_path = csv_output_path  # Update path to CSV
else:
    df = pd.read_csv(interaction_matrix_path, index_col=0)


valid_phages = set([f.split('.')[0] for f in os.listdir(dna_base+"/phage") if f.
 ↪endswith('.txt')])
valid_hosts = set([f.split('.')[0] for f in os.listdir(pro_base+"/bacteria") if␣
 ↪f.endswith('.txt')])


#filter unused interaction since matrix to large
df = df.loc[df.index.intersection(valid_hosts), df.columns.
 ↪intersection(valid_phages)]

phages = df.columns.tolist()
hosts = df.index.tolist()
print(len(phages))
print(len(hosts))
```

```
105
200
```

3 Prepare data as list of (phage, host, label)

[20]:
```
all_data = []

for p in phages:
    for h in hosts:
        label = df.loc[h, p]
        if pd.isna(label):
            continue  # Skip missing values
        binary_label = 1 if label >= 1 else 0
        all_data.append([p, h, binary_label])
```

3.0.1 Model Training and Evaluation (with Metrics)

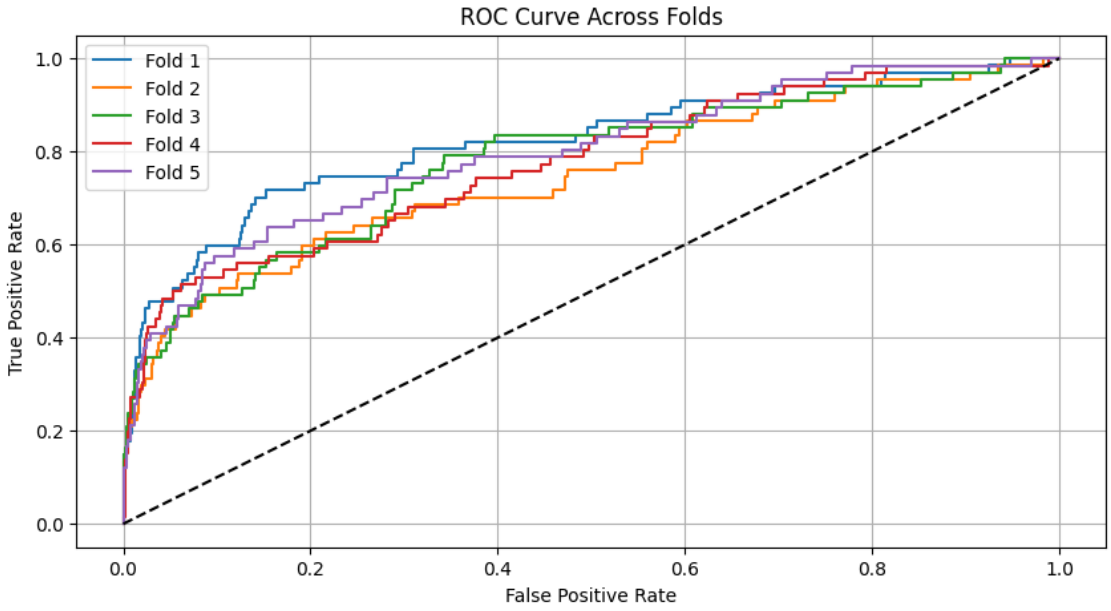
[22]:
```
results_all = []
fprs, tprs, precisions, recalls = [], [], [], []
# Your cross-validation setup:
kf = StratifiedKFold(n_splits=5, random_state=1, shuffle=True)
labels = [row[2] for row in all_data]

for fold, (train_idx, val_idx) in enumerate(kf.split(all_data, labels)):
    print(f"Fold {fold+1}")

    train_set = [all_data[i] for i in train_idx]
    val_set = [all_data[i] for i in val_idx]

    train_phages = [x[0] for x in train_set]
    train_hosts = [x[1] for x in train_set]
    train_labels = [x[2] for x in train_set]

    val_phages = [x[0] for x in val_set]
    val_hosts = [x[1] for x in val_set]
    val_labels = [x[2] for x in val_set]

    X_phage_dna_tr, X_host_dna_tr, X_phage_pro_tr, X_host_pro_tr, y_train =␣
 ↪obtain_features(
        train_phages, train_hosts, train_labels, dna_base, pro_base)
    X_phage_dna_val, X_host_dna_val, X_phage_pro_val, X_host_pro_val, y_val =␣
 ↪obtain_features(
        val_phages, val_hosts, val_labels, dna_base, pro_base)

    X_train_combined = combine_features(X_phage_dna_tr, X_host_dna_tr,␣
 ↪X_phage_pro_tr, X_host_pro_tr)
    X_val_combined = combine_features(X_phage_dna_val, X_host_dna_val,␣
 ↪X_phage_pro_val, X_host_pro_val)

    imbalance = sum([1 for i in y_train if i==1]) / sum([1 for i in y_train if␣
 ↪i==0])

    xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss')

    # Define hyperparameter grid
    param_grid = {
        'n_estimators': [100, 200, 250],
        'max_depth': [5, 7, None],
        'learning_rate': [0.1, 0.3],
        'scale_pos_weight': [1, 2, 1/imbalance]
    }
```

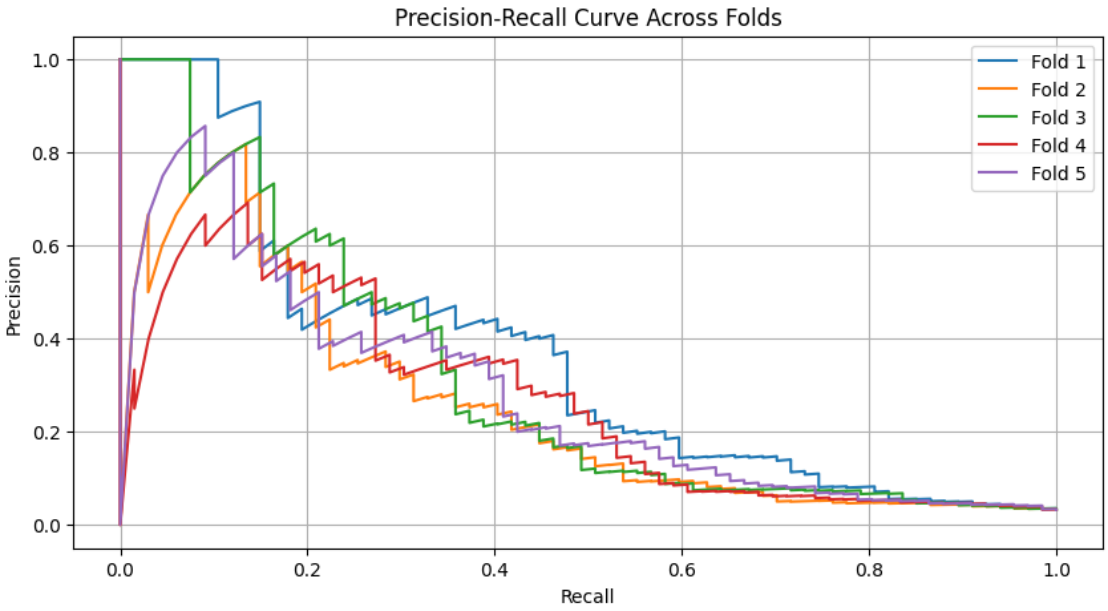
```
    # Grid search
    grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid,␣
↪scoring='average_precision', cv=5, n_jobs=-1, verbose=1)
    grid_search.fit(X_train_combined, y_train)
    best_model = grid_search.best_estimator_
    y_pred_prob = best_model.predict_proba(X_val_combined)[:, 1]


    #model = XGBClassifier(
    #    scale_pos_weight=1/imbalance,
    #    learning_rate=0.3,
    #    n_estimators=250,
    #    max_depth=7,
    #    use_label_encoder=False,
    #    eval_metric='logloss',
    #    n_jobs=8
    #)
    #model.fit(X_train_combined, y_train)
    #y_pred_prob = model.predict_proba(X_val_combined)[:,1]

    fold_metrics = scores(y_val, y_pred_prob)
    results_all.append(fold_metrics[:7])  # Save base metrics
    fprs.append(fold_metrics[7])
    tprs.append(fold_metrics[8])
    precisions.append(fold_metrics[9])
    recalls.append(fold_metrics[10])

    print(f"Fold {fold+1} | AUPR: {fold_metrics[0]:.4f}, AUC: {fold_metrics[1]:.
↪4f}, F1: {fold_metrics[2]:.4f}, Acc: {fold_metrics[3]:.4f}")
```

```
Fold 1
Fitting 5 folds for each of 54 candidates, totalling 270 fits
Fold 1 | AUPR: 0.3544, AUC: 0.8206, F1: 0.2558, Acc: 0.9680
Fold 2
Fitting 5 folds for each of 54 candidates, totalling 270 fits
Fold 2 | AUPR: 0.2418, AUC: 0.7489, F1: 0.2410, Acc: 0.9685
Fold 3
Fitting 5 folds for each of 54 candidates, totalling 270 fits
Fold 3 | AUPR: 0.3068, AUC: 0.7775, F1: 0.2588, Acc: 0.9685
Fold 4
Fitting 5 folds for each of 54 candidates, totalling 270 fits
Fold 4 | AUPR: 0.2543, AUC: 0.7754, F1: 0.2439, Acc: 0.9690
Fold 5
Fitting 5 folds for each of 54 candidates, totalling 270 fits
Fold 5 | AUPR: 0.2694, AUC: 0.7963, F1: 0.2727, Acc: 0.9680
```

3.0.2 Plotting Metrics Across Folds

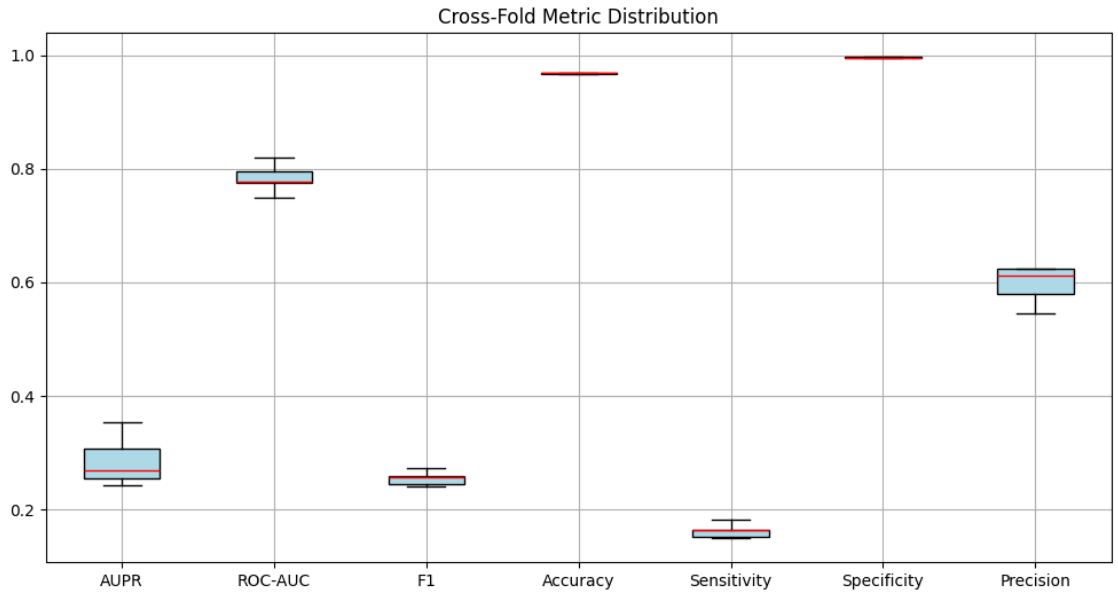
[23]:
```
# Plot ROC Curves
plt.figure(figsize=(10, 5))
for i in range(len(fprs)):
    plt.plot(fprs[i], tprs[i], label=f'Fold {i+1}')
plt.plot([0, 1], [0, 1], 'k--')
plt.title("ROC Curve Across Folds")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.grid()
plt.show()

# Plot PR Curves
plt.figure(figsize=(10, 5))
for i in range(len(precisions)):
    plt.plot(recalls[i], precisions[i], label=f'Fold {i+1}')
plt.title("Precision-Recall Curve Across Folds")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.legend()
plt.grid()
plt.show()
```





3.0.3 Summary Boxplot of All Metrics

[24]:
```
metric_names = ["AUPR", "ROC-AUC", "F1", "Accuracy", "Sensitivity",␣
 ↪"Specificity", "Precision"]
results_array = np.array(results_all)

plt.figure(figsize=(12, 6))
plt.boxplot(results_array, labels=metric_names, patch_artist=True,
            boxprops=dict(facecolor='lightblue', color='black'),
            medianprops=dict(color='red'),
            whiskerprops=dict(color='black'))
plt.title("Cross-Fold Metric Distribution")
plt.grid()
plt.show()
```



3.0.4 Fold-wise Metric Table

[25]:
```
results_df = pd.DataFrame(results_all, columns=metric_names)
results_df.index = [f"Fold {i+1}" for i in range(len(results_all))]
display(results_df)

print("Mean Metrics:")
display(results_df.mean())
```

```
           AUPR   ROC-AUC        F1  Accuracy  Sensitivity  Specificity  \
Fold 1  0.354439  0.820641  0.255814  0.968032     0.164179     0.995866
Fold 2  0.241840  0.748893  0.240964  0.968516     0.149254     0.996898
Fold 3  0.306834  0.777524  0.258824  0.968516     0.164179     0.996381
Fold 4  0.254304  0.775429  0.243902  0.969015     0.151515     0.996899
Fold 5  0.269388  0.796320  0.272727  0.968016     0.181818     0.994832

        Precision
Fold 1   0.578947
Fold 2   0.625000
Fold 3   0.611111
Fold 4   0.625000
Fold 5   0.545455

Mean Metrics:

AUPR           0.285361
ROC-AUC        0.783761
F1             0.254446
Accuracy       0.968419
```

```
Sensitivity     0.162189
Specificity     0.996175
Precision       0.597103
dtype: float64
```

3.1 Random Forest Model with Grid SearchCV

[27]:
```
rf_results_all = []
rf_fprs, rf_tprs, rf_precisions, rf_recalls = [], [], [], []
kf = StratifiedKFold(n_splits=5, random_state=1, shuffle=True)

rf_param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'max_features': ['auto', 'sqrt', 'log2'],
    'bootstrap': [True, False]
}
labels = [row[2] for row in all_data]

for fold, (train_idx, val_idx) in enumerate(kf.split(all_data, labels)):
    print(f"RF Fold {fold+1}")

    train_set = [all_data[i] for i in train_idx]
    val_set = [all_data[i] for i in val_idx]

    train_phages = [x[0] for x in train_set]
    train_hosts = [x[1] for x in train_set]
    train_labels = [x[2] for x in train_set]

    val_phages = [x[0] for x in val_set]
    val_hosts = [x[1] for x in val_set]
    val_labels = [x[2] for x in val_set]

    X_phage_dna_tr, X_host_dna_tr, X_phage_pro_tr, X_host_pro_tr, y_train =␣
 ↪obtain_features(
        train_phages, train_hosts, train_labels, dna_base, pro_base)
    X_phage_dna_val, X_host_dna_val, X_phage_pro_val, X_host_pro_val, y_val =␣
 ↪obtain_features(
        val_phages, val_hosts, val_labels, dna_base, pro_base)

    X_train_combined = combine_features(X_phage_dna_tr, X_host_dna_tr,␣
 ↪X_phage_pro_tr, X_host_pro_tr)
    X_val_combined = combine_features(X_phage_dna_val, X_host_dna_val,␣
 ↪X_phage_pro_val, X_host_pro_val)
```

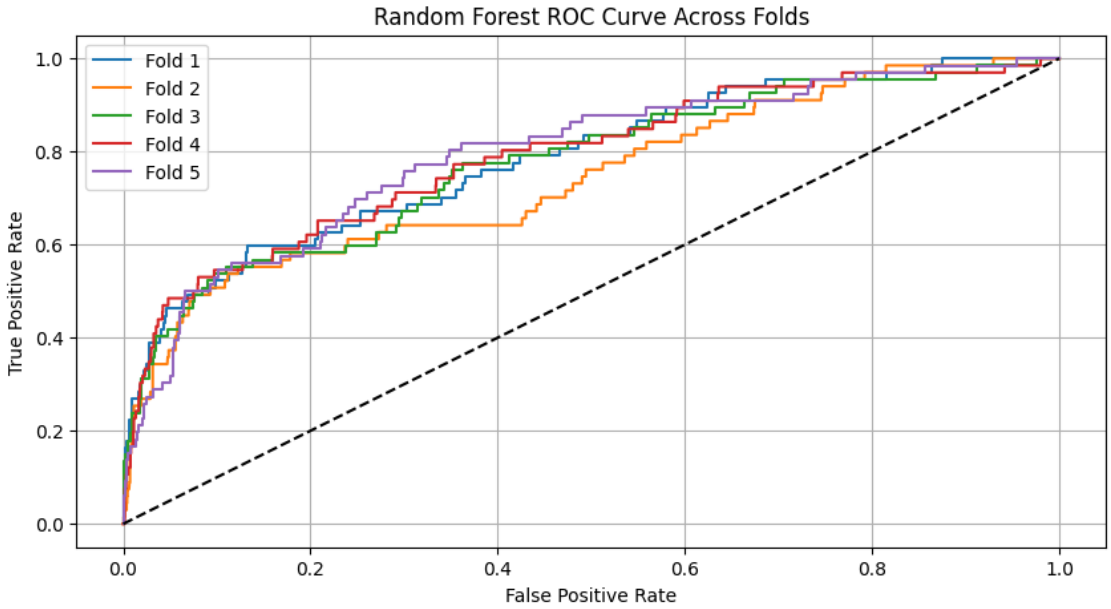
```
    class_weights = compute_class_weight(class_weight='balanced', classes=np.
↪array([0, 1]), y=y_train)
    class_weight_dict = {0: class_weights[0], 1: class_weights[1]}

    rf_base = RandomForestClassifier(class_weight=class_weight_dict,␣
↪random_state=42, n_jobs=-1)
    grid_search = GridSearchCV(estimator=rf_base, param_grid=rf_param_grid,
                               scoring='f1', cv=3, n_jobs=-1, verbose=0)
    grid_search.fit(X_train_combined, y_train)
    best_rf = grid_search.best_estimator_

    y_pred_prob = best_rf.predict_proba(X_val_combined)[:, 1]

    rf_fold_metrics = scores(y_val, y_pred_prob)
    rf_results_all.append(rf_fold_metrics[:7])
    rf_fprs.append(rf_fold_metrics[7])
    rf_tprs.append(rf_fold_metrics[8])
    rf_precisions.append(rf_fold_metrics[9])
    rf_recalls.append(rf_fold_metrics[10])

    print(f"Fold {fold+1} | AUPR: {rf_fold_metrics[0]:.4f}, AUC:␣
↪{rf_fold_metrics[1]:.4f}, "
          f"F1: {rf_fold_metrics[2]:.4f}, Acc: {rf_fold_metrics[3]:.4f}")
```

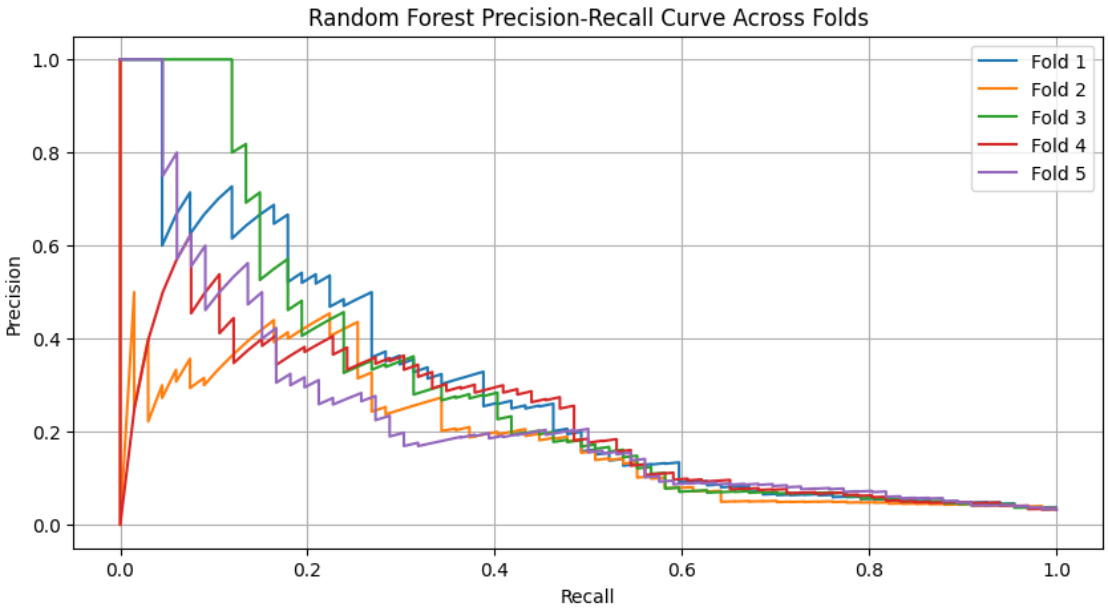
```
RF Fold 1
Fold 1 | AUPR: 0.2833, AUC: 0.7858, F1: 0.3146, Acc: 0.9391
RF Fold 2
Fold 2 | AUPR: 0.1777, AUC: 0.7437, F1: 0.2703, Acc: 0.9460
RF Fold 3
Fold 3 | AUPR: 0.2907, AUC: 0.7751, F1: 0.3000, Acc: 0.9510
RF Fold 4
Fold 4 | AUPR: 0.2140, AUC: 0.7885, F1: 0.3452, Acc: 0.9450
RF Fold 5
Fold 5 | AUPR: 0.2258, AUC: 0.7930, F1: 0.2414, Acc: 0.9560
```

3.1.1 Random Forest ROC & PR Curves

[28]:
```
plt.figure(figsize=(10, 5))
for i in range(len(rf_fprs)):
    plt.plot(rf_fprs[i], rf_tprs[i], label=f'Fold {i+1}')
plt.plot([0, 1], [0, 1], 'k--')
plt.title("Random Forest ROC Curve Across Folds")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.grid()
plt.show()
```

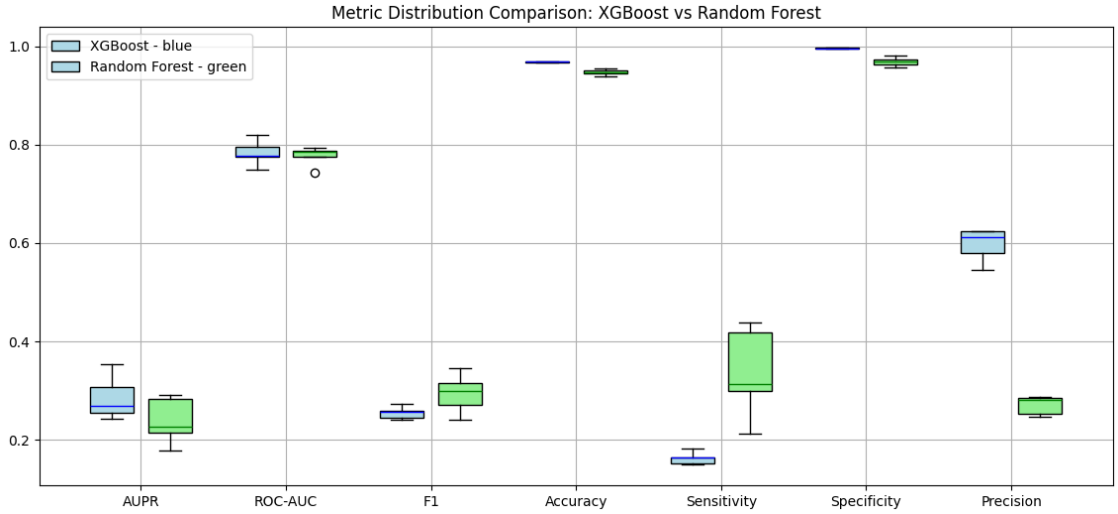
```
plt.figure(figsize=(10, 5))
for i in range(len(rf_precisions)):
    plt.plot(rf_recalls[i], rf_precisions[i], label=f'Fold {i+1}')
plt.title("Random Forest Precision-Recall Curve Across Folds")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.legend()
plt.grid()
plt.show()
```





3.1.2 Metric Boxplot: XGBoost vs Random Forest

[29]:
```
all_metric_names = ["AUPR", "ROC-AUC", "F1", "Accuracy", "Sensitivity",␣
 ↪"Specificity", "Precision"]

plt.figure(figsize=(14, 6))
plt.boxplot([np.array(results_all)[:,i] for i in range(7)],
            positions=np.arange(1, 8) - 0.2, widths=0.3, patch_artist=True,
            boxprops=dict(facecolor='lightblue'),␣
 ↪medianprops=dict(color='blue'), labels=all_metric_names)

plt.boxplot([np.array(rf_results_all)[:,i] for i in range(7)],
            positions=np.arange(1, 8) + 0.2, widths=0.3, patch_artist=True,
            boxprops=dict(facecolor='lightgreen'),␣
 ↪medianprops=dict(color='green'))

plt.legend(['XGBoost - blue', 'Random Forest - green'])
plt.title("Metric Distribution Comparison: XGBoost vs Random Forest")
plt.grid()
plt.xticks(np.arange(1, 8), all_metric_names)
plt.show()
```



3.1.3 Random Forest Fold-wise Metrics

[30]:
```
rf_df = pd.DataFrame(rf_results_all, columns=all_metric_names)
rf_df.index = [f"Fold {i+1}" for i in range(len(rf_results_all))]
display(rf_df)

print("Random Forest - Mean Metrics:")
display(rf_df.mean())
```

	AUPR	ROC-AUC	F1	Accuracy	Sensitivity	Specificity	Precision
Fold 1	0.283325	0.785815	0.314607	0.939061	0.417910	0.957106	0.252252
Fold 2	0.177674	0.743683	0.270270	0.946027	0.298507	0.968459	0.246914
Fold 3	0.290701	0.775147	0.300000	0.951024	0.313433	0.973113	0.287671
Fold 4	0.214031	0.788474	0.345238	0.945027	0.439394	0.962274	0.284314
Fold 5	0.225815	0.793015	0.241379	0.956022	0.212121	0.981395	0.280000

Random Forest - Mean Metrics:

AUPR	0.238309
ROC-AUC	0.777227
F1	0.294299
Accuracy	0.947432
Sensitivity	0.336273
Specificity	0.968469

```
Precision      0.270230
dtype: float64
```

3.1.4 Side-by-side Mean Metrics (XGBoost vs RF)

[31]:
```
xgb_mean = pd.DataFrame([np.mean(results_all, axis=0)],␣
 ↪columns=all_metric_names, index=["XGBoost"])
rf_mean = pd.DataFrame([np.mean(rf_results_all, axis=0)],␣
 ↪columns=all_metric_names, index=["Random Forest"])

display(pd.concat([xgb_mean, rf_mean]))
```

```
                  AUPR   ROC-AUC        F1  Accuracy  Sensitivity  \
XGBoost       0.285361  0.783761  0.254446  0.968419     0.162189
Random Forest 0.238309  0.777227  0.294299  0.947432     0.336273

              Specificity  Precision
XGBoost          0.996175   0.597103
Random Forest    0.968469   0.270230
```

4 XGBoost with LOGO

[31]:
```
# Assuming:
# all_data = [(phage, host, label), ...]
# hosts = list of host IDs
# phages = list of phage IDs

# Create groups based on host for LOGO
groups = []
for sample in all_data:
    phage, host, label = sample
    group_id = hosts.index(host)  # group by host
    groups.append(group_id)

print(f"Number of unique groups (hosts): {len(set(groups))}")
```

```
Number of unique groups (hosts): 65
```

[32]:
```
logo = LeaveOneGroupOut()
cpus = 8

# For collecting predictions and labels
scores_all = []
label_list = []

# Progress bar for number of groups
pbar = tqdm(total=len(set(groups)))
```

```
for fold, (train_idx, val_idx) in enumerate(logo.split(all_data,␣
↪groups=groups)):
    # Prepare data
    train_set = [all_data[i] for i in train_idx]
    val_set = [all_data[i] for i in val_idx]

    train_phages = [x[0] for x in train_set]
    train_hosts = [x[1] for x in train_set]
    train_labels = [x[2] for x in train_set]

    val_phages = [x[0] for x in val_set]
    val_hosts = [x[1] for x in val_set]
    val_labels = [x[2] for x in val_set]

    # Obtain features
    X_phage_dna_tr, X_host_dna_tr, X_phage_pro_tr, X_host_pro_tr, y_train =␣
↪obtain_features(train_phages, train_hosts, train_labels, dna_base, pro_base)
    X_phage_dna_val, X_host_dna_val, X_phage_pro_val, X_host_pro_val, y_val =␣
↪obtain_features(val_phages, val_hosts, val_labels, dna_base, pro_base)


    # Combine features for train and val
    X_train_combined = combine_features(X_phage_dna_tr, X_host_dna_tr,␣
↪X_phage_pro_tr, X_host_pro_tr)
    X_val_combined = combine_features(X_phage_dna_val, X_host_dna_val,␣
↪X_phage_pro_val, X_host_pro_val,)

    # Class imbalance handling
    pos = sum(y_train)
    neg = len(y_train) - pos
    scale_pos_weight = neg / pos if pos > 0 else 1

    # Train
    model = XGBClassifier(
        scale_pos_weight=scale_pos_weight,
        learning_rate=0.3,
        n_estimators=250,
        max_depth=7,
        use_label_encoder=False,
        eval_metric='logloss',
        n_jobs=cpus
    )
    model.fit(X_train_combined, y_train)

    # Predict
    y_pred_prob = model.predict_proba(X_val_combined)[:, 1]
    scores_all.append(y_pred_prob)
```

```
    label_list.append(y_val)

    pbar.update(1)

pbar.close()

# Concatenate all predictions and labels
y_pred_all = np.concatenate(scores_all)
y_true_all = np.concatenate(label_list)

# Now evaluate final performance
final_metrics = scores(y_true_all, y_pred_all)

print("\nFinal LOGO metrics across all folds:")
print(f"AUPR: {final_metrics[0]:.4f}, AUC: {final_metrics[1]:.4f}, F1:␣
 ↪{final_metrics[2]:.4f}, Accuracy: {final_metrics[3]:.4f}")
```

```
  0%|          | 0/65 [00:00<?, ?it/s]
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[32], line 25
     22 val_labels = [x[2] for x in val_set]
     24 # Obtain features
---> 25 X_phage_dna_tr, X_host_dna_tr, X_phage_pro_tr, X_host_pro_tr, y_train =
 ↪obtain_features(train_phages, train_hosts, train_labels, dna_base, pro_base)
     26 X_phage_dna_val, X_host_dna_val, X_phage_pro_val, X_host_pro_val, y_val
 ↪= obtain_features(val_phages, val_hosts, val_labels, dna_base, pro_base)
     29 # Combine features for train and val

Cell In[19], line 5, in obtain_features(phage_list, host_list, labels, dna_base
 ↪pro_base)
      3 X_phage_pro, X_host_pro = [], []
      4 for p, h in zip(phage_list, host_list):
----> 5     X_phage_dna.
 ↪append(load_feature_vector(os.path.join(dna_base, 'phage', f'{p}.txt')))
      6     X_host_dna.append(load_feature_vector(os.path.join(dna_base,␣
 ↪'bacteria', f'{h}.txt')))
      7     X_phage_pro.append(load_feature_vector(os.path.join(pro_base,␣
 ↪'phage', f'{p}.txt')))

Cell In[5], line 2, in load_feature_vector(file_path)
      1 def load_feature_vector(file_path):
----> 2     return np.loadtxt(file_path)
```

```
File D:\Bachelorarbeit\Prediction Notebooks\.
 ↪venv2\lib\site-packages\numpy\lib\_npyio_impl.py:1395, in loadtxt(fname,␣
 ↪dtype, comments, delimiter, converters, skiprows, usecols, unpack, ndmin,␣
 ↪encoding, max_rows, quotechar, like)
   1392 if isinstance(delimiter, bytes):
   1393     delimiter = delimiter.decode('latin1')
-> 1395 arr = _read(fname, dtype=dtype, comment=comment, delimiter=delimiter,
   1396             converters=converters, skiplines=skiprows, usecols=usecols,
   1397             unpack=unpack, ndmin=ndmin, encoding=encoding,
   1398             max_rows=max_rows, quote=quotechar)
   1400 return arr

File D:\Bachelorarbeit\Prediction Notebooks\.
 ↪venv2\lib\site-packages\numpy\lib\_npyio_impl.py:1022, in _read(fname,␣
 ↪delimiter, comment, quote, imaginary_unit, usecols, skiplines, max_rows,␣
 ↪converters, ndmin, unpack, dtype, encoding)
   1020     fname = os.fspath(fname)
   1021 if isinstance(fname, str):
-> 1022     fh = np.lib._datasource.open(fname, 'rt', encoding=encoding)
   1023     if encoding is None:
   1024         encoding = getattr(fh, 'encoding', 'latin1')

File D:\Bachelorarbeit\Prediction Notebooks\.
 ↪venv2\lib\site-packages\numpy\lib\_datasource.py:192, in open(path, mode,␣
 ↪destpath, encoding, newline)
    155 """
    156 Open `path` with `mode` and return the file object.
    157
   (…)
    188
    189 """
    191 ds = DataSource(destpath)
--> 192 return ds.open(path, mode, encoding=encoding, newline=newline)

File D:\Bachelorarbeit\Prediction Notebooks\.
 ↪venv2\lib\site-packages\numpy\lib\_datasource.py:529, in DataSource.open(self␣
 ↪path, mode, encoding, newline)
    526     return _file_openers[ext](found, mode=mode,
    527                               encoding=encoding, newline=newline)
    528 else:
--> 529     raise FileNotFoundError(f"{path} not found.")

FileNotFoundError: ../ordina_dataset_features/
 ↪dna_features_ordinal_data\phage\001-023.txt not found.
```

[ ]:

[ ]: