PPI Prediction Project

January 30, 2024

1 Protein Protein Interaction (PPI) Prediciton Project

The upcoming project revolves around predicting Protein-Protein Interactions (PPI). Our objective is to extract data from our Timbal dataset, which currently includes UniProt target information but lacks UniProt partner details.

The project will involve taking the dataset that indeed contains both UniProt partner and UniProt target information. We will segregate molecules and their corresponding SMILES values that share the same UniProt target. This subset of data will be used for subsequent predictions, and new datasets will be constructed to encompass all such molecules.

Subsequently, we will visualize our datasets and analyze the data distribution to gain a deeper understanding.

Following that, we plan to develop three models: - Random Forest Multiclass Classifier Model - XGBoost Multiclass Classifier Model - GraphConvModel Multiclass Classifier Model

The first two models will be implemented on two types of dataframes for each dataset. The first dataframe will utilize feature augmentation techniques with RDKitDescriptors, while the second will employ Morgan Fingerprint.

The last model will be implemented using the DeepChem library.

For each dataset, we will select the most suitable model. Finally, we will predict the UniProt partners for each dataset using their corresponding models. Following the prediction phase, we will comataframe containing all the predicted data— of interest.

- RDKit library official website : https://www.rdkit.org/docs/index.html
- DeepChem library official website : https://deepchem.io/

1.1 Import Libraries for the Project

[854]:

```
import os
import pickle  # In order to save DataFrame dictionary

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from joblib import dump, load  # For saving & loading trained models

from sklearn.metrics import (
    roc_curve,
    auc,
    roc_auc_score,
    make_scorer,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    confusion_matrix,
    classification_report,
)
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import label_binarize
from sklearn.utils.class_weight import compute_sample_weight,␣
 ↪compute_class_weight
from sklearn.model_selection import train_test_split, GridSearchCV,␣
 ↪StratifiedKFold
from sklearn.utils import resample  # For executing bootstrap to get more␣
 ↪accurate roc_auc_score when testing best models

import xgboost as xgb

from rdkit import Chem
from rdkit.Chem import AllChem, PandasTools, Descriptors, rdmolops

import deepchem as dc
from deepchem.feat import RDKitDescriptors
from deepchem.models import GraphConvModel
from deepchem.hyper import GridHyperparamOpt, HyperparamOpt
from deepchem.splits.splitters import RandomGroupSplitter
from deepchem.trans import undo_transforms
from deepchem.trans.transformers import BalancingTransformer

import warnings
warnings.filterwarnings('ignore')
```

1.1.1 Basic Helper Functions

[117]:

```
def PRINT(text) -> None: print(f"{'~'*80}\n{text}\n{'~'*80}")

def is_numeric(value):
    """
    Checks if a given value can be converted to a float, indicating numeric nature.

    Parameters:
    - value: Any value to check.
    """
    try:
        float(value)
        return True
    except (ValueError, TypeError):
        return False

def print_dict_meaningful(dictionary):
    """
    Prints key-value pairs of a dictionary, formatting numeric values to 3 decimal places.

    Parameters:
    - dictionary: A dictionary to print.
    """
    for key, value in dictionary.items():
        if is_numeric(value):
            formatted_value = "{:.3f}".format(float(value))
        else:
            formatted_value = value
        print(f'{key}: {formatted_value}')

def plot_uniprot_numeric_label_frequency(df, UniProt) -> None:
    """
    Plots a countplot of NumericUniProtTargetLabels for a specific UniProt ID.

    Parameters:
    - df: DataFrame containing the data.
    - UniProt: UniProt ID for which the countplot is generated.e
    """
    plt.figure(figsize=(10, 6))
    sns.countplot(x='NumericUniProtTargetLabels', data=df)
    plt.title(f'Countplot of NumericUniProtTargetLabels for UniProt {UniProt}')
    plt.xlabel('NumericUniProtTargetLabels')
    plt.ylabel('Frequency')
    plt.show()
```

1.2 Preparing Datasets for Predictive Modeling

1.2.1 Load Required Datasets

[3]:
```
pwd
```

[3]:
```
'C:\\Users\\gavvi\\Desktop\\Programming\\GitHub\\DeepLearningResearchStarship\\P
roject 4 Protein Relationship Prediction'
```

[4]:
```
pred_dataset_path = "data/dataset_for_prediction.csv"
ChEMBL_integrin_dataset_path = "data/ChEMBL_Integrins.csv"
```

[5]:
```
pred_df = pd.read_csv(pred_dataset_path)

pred_df.head(5)
```

[5]:
```
                                              smiles uniprot_id1
0  OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cccc2Cl)cc1)NC…      P13612
1  C\C=C\[C@@H](CC(=O)O)NC(=O)C[C@@H](CC(C)C)NC(=…      P05556
2  CN1[C@@H](CCCN=C(N)N)C(=O)NCC(=O)N[C@@H](CC(=O…      P05106
3   OC(=O)C(CNC(=O)CCCCc1ccc2CCCNc2n1)c3cnc4ccccc4c3      P05106
4  OC(=O)C[C@H](NC(=O)CN1CCC[C@@H](CCC2CCNCC2)C1=…      P05106
```

Next, we aim to rename the column *uniprot_id1* to *uniprot_id*. The rationale behind this decision is that we intend to search for this value in the *ChEMBL* data frame within both the *uniprot1* and *uniprot2* columns. To minimize confusion, we will rename this column

[6]:
```
pred_df = pred_df.rename(columns={'uniprot_id1':'uniprot_id'})

PRINT(f'Renamed column name: {pred_df.columns[1]}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Renamed column name: uniprot_id
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[15]:
```
chmbl_df = pd.read_csv(ChEMBL_integrin_dataset_path)

chmbl_df.head(5)
```

[15]:
```
                              Canonical SMILES(RDKit)         Target Pref Name  \
0  N=C(N)NCCC[C@H](NC(=O)[C@H](CCCNC(=N)N)NC(=O)[…  Integrin alpha-4/beta-7
1  N=C(N)NCCC[C@H](NC(=O)CCCC[C@@H]1SC[C@@H]2NC(=…  Integrin alpha-4/beta-7
2  N#Cc1ccc(-c2ccc(C[C@H](NC(=O)[C@H](CCCNC(=N)N)…  Integrin alpha-4/beta-7
3  N=C(N)NCCC[C@H](NC(=O)CCCC[C@@H]1SC[C@@H]2NC(=…  Integrin alpha-4/beta-7
4  N=C(N)NCCC[C@H](NC(=O)CCCC[C@@H]1SC[C@@H]2NC(=…  Integrin alpha-4/beta-7

       Organism UniProt1 UniProt2 UniProt3 UniProt4 UniProt5
0  Mus musculus   Q00651   P26011      NaN      NaN      NaN
1  Mus musculus   Q00651   P26011      NaN      NaN      NaN
```

```
2  Mus musculus   Q00651   P26011      NaN      NaN      NaN
3  Mus musculus   Q00651   P26011      NaN      NaN      NaN
4  Mus musculus   Q00651   P26011      NaN      NaN      NaN
```

1.2.2 Generate Unique Datasets

Prechecks

```
[22]: unique_proteins = pred_df["uniprot_id1"].unique()
```

```
[26]: PRINT(f"The unique proteins we want to predict their partners in the PPI are :
        ↪\n {unique_proteins}\n\nThe are total {len(unique_proteins)} such proteins")
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The unique proteins we want to predict their partners in the PPI are :
 ['P13612' 'P05556' 'P05106' 'P05107' 'P08648' 'P17301']

The are total 6 such proteins
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Datasets Generation Phase The next step is to generate six datasets, each for the protein for which we intend to create a deep learning model to predict its companion in PPI (i.e., the second UniProt_id).

The way we are going to achieve this is by taking each unique *UniProt_id* value, searching for all the rows in the *ChEMBL* data frame we loaded from the previous project, where that *UniProt_id* value is one of their *UniProt_id{i}* columns, where i [1,5].

Each dataset will contain all the molecules' *SMILES* values, with both *UniProt_ids* forming the connection.

From these datasets, we will proceed to train our model. Thus, we can provide the unique SMILES value along with the UniProt_id to the model, and it will predict its partner.

```
[69]: protein_dataframes = {}

      for protein in unique_proteins:
          # Initialize an empty list to store rows for the current protein
          rows_for_protein = []

          # Iterate over each row in the ChEMBL DataFrame
          for index, row in chmbl_df.iterrows():
              # Check if the current protein is present in any of the UniProt columns
              if protein in row[['UniProt1', 'UniProt2', 'UniProt3', 'UniProt4',␣
        ↪'UniProt5']].values:
                  # Determine the correct order (UniProt1 and UniProt2) in the new␣
        ↪data frame
                  if row['UniProt1'] == protein:
                      relevant_info = [row['Canonical SMILES(RDKit)'],␣
        ↪row['UniProt1'], row['UniProt2']]
```

```
            elif row['UniProt2'] == protein:
                relevant_info = [row['Canonical SMILES(RDKit)'],␣
↪row['UniProt2'], row['UniProt1']]
            else:
                relevant_info = []

            if relevant_info:
                rows_for_protein.append(relevant_info)

    if rows_for_protein:
        protein_dataframes[protein] = pd.DataFrame(rows_for_protein,␣
↪columns=['SMILES', 'UniProt1', 'UniProt2'])
```

[49]:

```
protein_dataframes['P13612']
```

[49]:

```
                                                  SMILES UniProt1 UniProt2
0     COc1ccccc1-c1ccc(C[C@H](NC(=O)c2ccccc2Cl)C(=O)…   P13612   P26010
1     Cc1ccccc1NC(=O)Nc1ccc(CC(=O)N2C[C@@H](F)C[C@H]…   P13612   P26010
2     CN(C)Cc1ccccc1-c1ccc(C[C@H](NC(=O)c2ccccc2Cl)C…   P13612   P26010
3     Cc1cccc(Cl)c1C(=O)N[C@@H](Cc1ccc(NC(=O)c2c(Cl)…   P13612   P26010
4     COc1cnn(C)c(=O)c1-c1ccc(C[C@H](NC(=O)c2c(C)noc…   P13612   P26010
…                                                  …      …      …
1969  CC(C)(C)[C@H]1CC[C@H](C[C@H](NC(=O)[C@@H]2CCC(…   P13612   P05556
1970  O=C(Nc1ccc(C[C@H](/N=c2\c(O)c(O)\c2=N/Cc2ccccc…   P13612   P05556
1971  N#Cc1cccc(S(=O)(=O)N2C[C@H](N3CCC(F)CC3)C[C@H]…   P13612   P05556
1972  CCCCS(=O)(=O)N[C@@H](Cc1ccc(OCCCCC2CCNCC2)cc1)…   P13612   P05556
1973  O=C(Cc1ccc2nc(-c3ccccc3)oc2c1)N1C[C@@H](F)C[C@…   P13612   P05556

[1974 rows x 3 columns]
```

Save the Data Frames Dictionary

[323]:

```
directory_path = 'obj'

# Save the dictionary to a file in the specified directory
with open(os.path.join(directory_path, 'data_frames_dictionary.pkl'), 'wb') as␣
↪file:
    pickle.dump(protein_dataframes, file)
```

Save the Generated Data Frame as CSV Files

[50]:

```
out_dir = 'unique UniProt csv files'
```

[51]:

```
for protein, df in protein_dataframes.items():
    try:
        # Generate csv file name with the desired format
        file_name = f'{protein}.csv'

        # Specify full path
        out_path = os.path.join(out_dir, file_name)

        # Save current data frame as csv file
        df.to_csv(out_path, index=False)

        PRINT(f'Saved data frame for {protein} as {file_name}')

    except Exception as e:
        PRINT(f'Error!\nVerify path name and the data')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Saved data frame for P13612 as P13612.csv
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Saved data frame for P05556 as P05556.csv
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Saved data frame for P05106 as P05106.csv
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Saved data frame for P05107 as P05107.csv
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Saved data frame for P08648 as P08648.csv
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Saved data frame for P17301 as P17301.csv
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.2.3 Visualize Distributions for each Data Frame

[60]:

```
PRINT(f'We have {len(protein_dataframes.items())} data frames to visualize␣
  ↪information about their data distributions')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
We have 6 data frames to visualize information about their data distributions
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[62]:

```
PRINT(f'UniProt_ids -> {unique_proteins}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
UniProt_ids -> ['P13612' 'P05556' 'P05106' 'P05107' 'P08648' 'P17301']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Helper Functions

Helper One-Hot-Encoding Function

[38]:

```
def one_hot_encoding(df):

    df_encoded = pd.get_dummies(df[['UniProt1', 'UniProt2']], prefix='',␣
 ↪prefix_sep='').astype(int)
    df_encoded = pd.concat([df[['SMILES']], df_encoded], axis=1)
    return df_encoded
```

Helper Visualization Function

[39]:

```
def visualize_dist(df, target_prot)-> None:
    # Melt the DataFrame to long format for Seaborn countplot
    df_melted = df.melt(var_name='Protein', value_name='Interaction Status')

    # Set the size of the plot
    sns.set(rc={'figure.figsize':(12, 8)})

    sns.set_context("notebook", rc={"lines.linewidth": 2.5})
    # Create a grouped count plot
    sns.countplot(x='Protein', hue='Interaction Status', palette=["lightgrey",␣
 ↪"skyblue"], data=df_melted)

    # Add labels and title
    plt.xlabel('Protein')
    plt.ylabel('Count')
    plt.title(f'PPI with -> {target_prot}')

    sns.despine()
    sns.set_theme(style="whitegrid")
    sns.despine(offset=10, trim=True)
    sns.set_context("notebook")
    plt.show()
```

Helper Column Filter Function

[5]:

```
def filter_proteins_list(df, columns_to_remove):

    filtered_columns = [col for col in df.columns if col not in␣
 ↪columns_to_remove]
    filtered_columns_list = list(filtered_columns)
    return filtered_columns
```

First Data Frame

[228]:

```
first_df = protein_dataframes[unique_proteins[0]]

first_df.head(2)
```

[228]:

```
                                              SMILES UniProt1 UniProt2
0  COc1ccccc1-c1ccc(C[C@H](NC(=O)c2ccccc2Cl)C(=O)…   P13612   P26010
1  Cc1ccccc1NC(=O)Nc1ccc(CC(=O)N2C[C@@H](F)C[C@H]…   P13612   P26010
```

Visualize Distribution

```
[234]: first_df_encoded = one_hot_encoding(first_df)
```

```
[235]: print(first_df_encoded.columns)
```

```
Index(['SMILES', 'P13612', 'P05556', 'P26010'], dtype='object')
```

```
[236]: first_df_encoded.head(3)
```

```
[236]:                                               SMILES  P13612  P05556  P26010
0  COc1ccccc1-c1ccc(C[C@H](NC(=O)c2ccccc2Cl)C(=O)…       1       0       1
1  Cc1ccccc1NC(=O)Nc1ccc(CC(=O)N2C[C@@H](F)C[C@H]…       1       0       1
2  CN(C)Cc1ccccc1-c1ccc(C[C@H](NC(=O)c2ccccc2Cl)C…       1       0       1
```

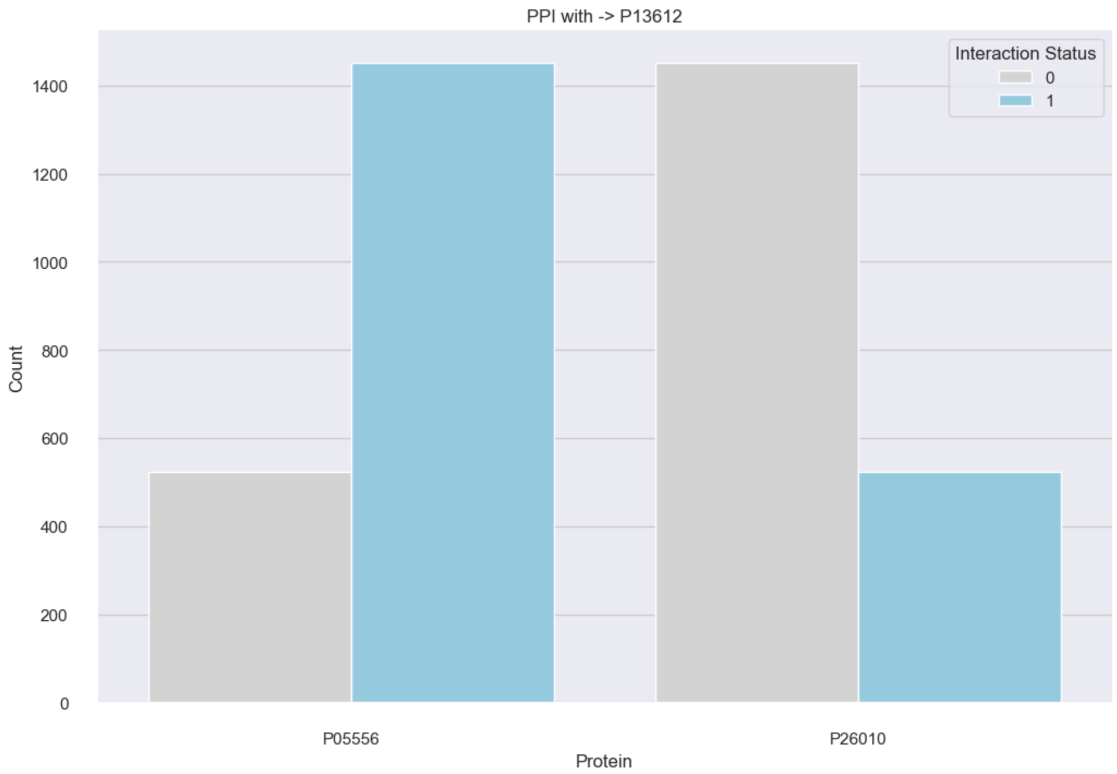
```
[237]: filtered_columns = filter_proteins_list(first_df_encoded, columns_to_remove =␣
         ↪['SMILES', 'P13612'])
       PRINT(f'Filtered columns -> {filtered_columns}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Filtered columns -> ['P05556', 'P26010']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[238]: temp_df_1 = first_df_encoded[filtered_columns]

       temp_df_1.head(2)
```

```
[238]:    P05556  P26010
0       0       1
1       0       1
```

```
[268]: visualize_dist(temp_df_1, unique_proteins[0])
```



As we can see from the histogram, `P05556` appears much more than `P26010` in the PPI with UniProt traget `P13612`

Explore the First Data Frame

[251]:
```
PRINT(f'The size of the data frame is -> {len(first_df)}')
print(f'Number of times P05556 appears -> {len(first_df[first_df["UniProt2"] ==␣
 ↪"P05556"])}')
print(f'Number of times P26010 appears -> {len(first_df[first_df["UniProt2"] ==␣
 ↪"P26010"])}')
print(f'Size check -> {({len(first_df)} == {(len(first_df[first_df["UniProt2"]␣
 ↪== "P05556"]) + len(first_df[first_df["UniProt2"] == "P26010"]))})}')

PRINT('Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The size of the data frame is -> 1974
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Number of times P05556 appears -> 1452
Number of times P26010 appears -> 522
Size check -> True
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Second Data Frame

```
[222]: second_df = protein_dataframes[unique_proteins[1]]

       second_df.head(2)
```

```
[222]:                                               SMILES UniProt1 UniProt2
       0  CC(C)(C)c1cc(Br)cc([C@H](CC(=O)O)NC(=O)CNC(=O)…   P05556   O75578
       1  O=C(NCc1ccccc1)NC[C@H](NC(=O)[C@@H]1CCCN1S(=O)…   P05556   P56199
```

Visualize Distribution

```
[223]: second_df_encoded = one_hot_encoding(second_df)
```

```
[252]: second_df_encoded.columns
```

```
[252]: Index(['SMILES', 'P05556', 'O75578', 'P05106', 'P06756', 'P08648', 'P13612',
              'P17301', 'P23229', 'P56199', 'Q13797'],
             dtype='object')
```

```
[224]: second_df_encoded.head(3)
```

```
[224]:                                               SMILES  P05556  O75578  P05106  \
       0  CC(C)(C)c1cc(Br)cc([C@H](CC(=O)O)NC(=O)CNC(=O)…       1       1       0
       1  O=C(NCc1ccccc1)NC[C@H](NC(=O)[C@@H]1CCCN1S(=O)…       1       0       0
       2  O=C(NCc1ccccc1)NC[C@H](NC(=O)[C@@H]1CCCN1S(=O)…       1       0       0

          P06756  P08648  P13612  P17301  P23229  P56199  Q13797
       0       0       0       0       0       0       0       0
       1       0       0       0       0       0       1       0
       2       0       0       0       0       0       1       0
```

```
[225]: filtered_columns = filter_proteins_list(second_df_encoded, columns_to_remove =␣
         ↪['SMILES', 'P05556'])
       PRINT(f'Filtered columns -> {filtered_columns}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Filtered columns -> ['O75578', 'P05106', 'P06756', 'P08648', 'P13612', 'P17301',
'P23229', 'P56199', 'Q13797']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

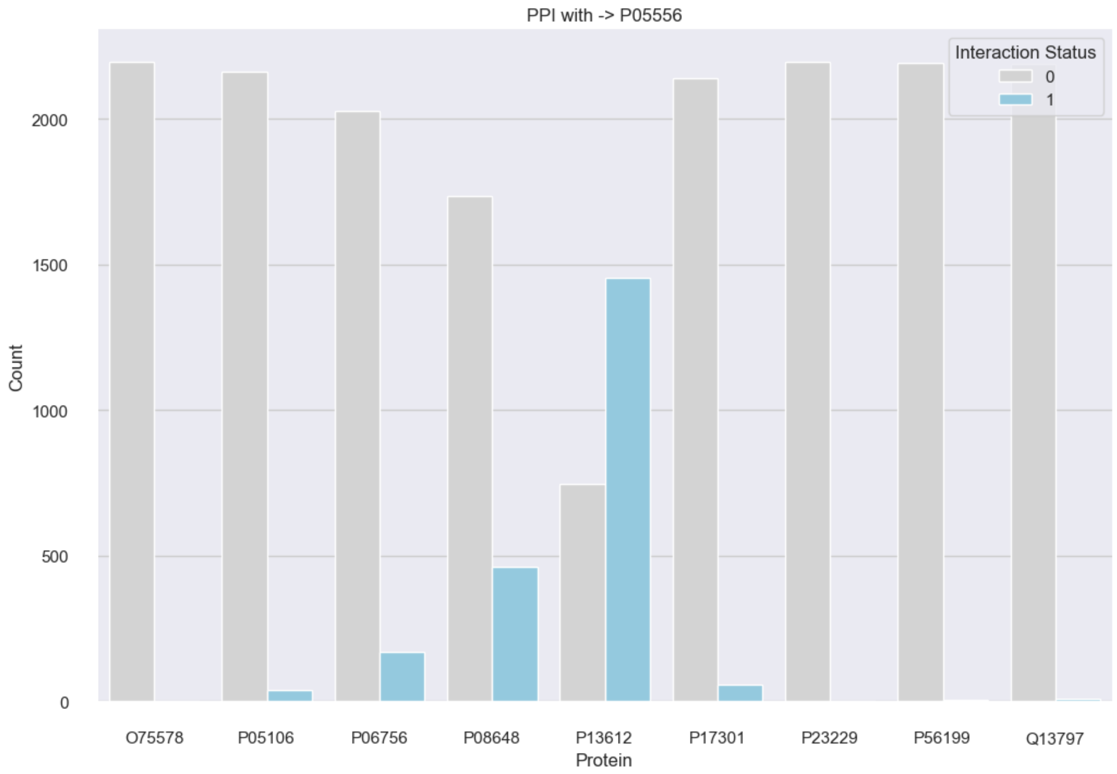
```
[226]: temp_df_2 = second_df_encoded[filtered_columns]

       temp_df_2.head(5)
```

```
[226]:    O75578  P05106  P06756  P08648  P13612  P17301  P23229  P56199  Q13797
       0       1       0       0       0       0       0       0       0       0
       1       0       0       0       0       0       0       0       1       0
       2       0       0       0       0       0       0       0       1       0
       3       0       0       0       0       0       0       0       1       0
```

```
4         0         0         0         0         0         0         0         1         0
```

[267]:

```
visualize_dist(temp_df_2, unique_proteins[1])
```



In the plot above, we observe that certain proteins, such as O75578 and P23229, have minimal occurrences in the PPI with P05106. In contrast, proteins like P13612 exhibit frequent appearances in the PPI with ’P05106.

Explore the Second Data Frame

[250]:

```
PRINT(f'The size of the data frame is -> {len(second_df)}')
print(f'Number of time O75578 appears -> {len(second_df[second_df["UniProt2"]␣
  ↪== "O75578"])}')
print(f'Number of time P23229 appears -> {len(second_df[second_df["UniProt2"]␣
  ↪== "P23229"])}')
print(f'Number of time P56199 appears -> {len(second_df[second_df["UniProt2"]␣
  ↪== "P56199"])}')
print(f'Number of time Q13797 appears -> {len(second_df[second_df["UniProt2"]␣
  ↪== "Q13797"])}')
print(f'Number of time P17301 appears -> {len(second_df[second_df["UniProt2"]␣
  ↪== "P17301"])}')
print(f'Number of time P05106 appears -> {len(second_df[second_df["UniProt2"]␣
  ↪== "P05106"])}')
```

```
print(f'Number of time P06756 appears -> {len(second_df[second_df["UniProt2"]␣
 ↪== "P06756"])}')
print(f'Number of time P08648 appears -> {len(second_df[second_df["UniProt2"]␣
 ↪== "P08648"])}')
print(f'Number of time P13612 appears -> {len(second_df[second_df["UniProt2"]␣
 ↪== "P13612"])}')

PRINT('Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The size of the data frame is -> 2197
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Number of time O75578 appears -> 1
Number of time P23229 appears -> 1
Number of time P56199 appears -> 6
Number of time Q13797 appears -> 10
Number of time P17301 appears -> 57
Number of time P05106 appears -> 37
Number of time P06756 appears -> 170
Number of time P08648 appears -> 463
Number of time P13612 appears -> 1452
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Third Data Frame

[263]:
```
third_df = protein_dataframes[unique_proteins[2]]

third_df.head(2)
```

[263]:
```
                                              SMILES UniProt1 UniProt2
0  CC(C)Oc1ccc(C(CC(=O)O)NC(=O)CCC(=O)Nc2ccc3c(c2…   P05106   P26006
1  COc1ccc(C(CC(=O)O)NC(=O)c2cccc(C(=O)Nc3ccc4c(c…   P05106   P26006
```

Visualize Distribution

[264]:
```
third_df_encoded = one_hot_encoding(third_df)
```

[265]:
```
third_df_encoded.columns
```

[265]:
```
Index(['SMILES', 'P05106', 'P05556', 'P06756', 'P08514', 'P17301', 'P26006'],
dtype='object')
```

[261]:
```
third_df_encoded.head(3)
```

[261]:
```
                                              SMILES  P05106  P05556  P06756  \
0  CC(C)Oc1ccc(C(CC(=O)O)NC(=O)CCC(=O)Nc2ccc3c(c2…       1       0       0
1  COc1ccc(C(CC(=O)O)NC(=O)c2cccc(C(=O)Nc3ccc4c(c…       1       0       0
```

```
2  COc1ccc(C(CC(=O)O)NC(=O)CCC(=O)Nc2ccc3c(c2)CNC…        1        0        0

   P08514  P17301  P26006
0       0       0       1
1       0       0       1
2       0       0       1
```

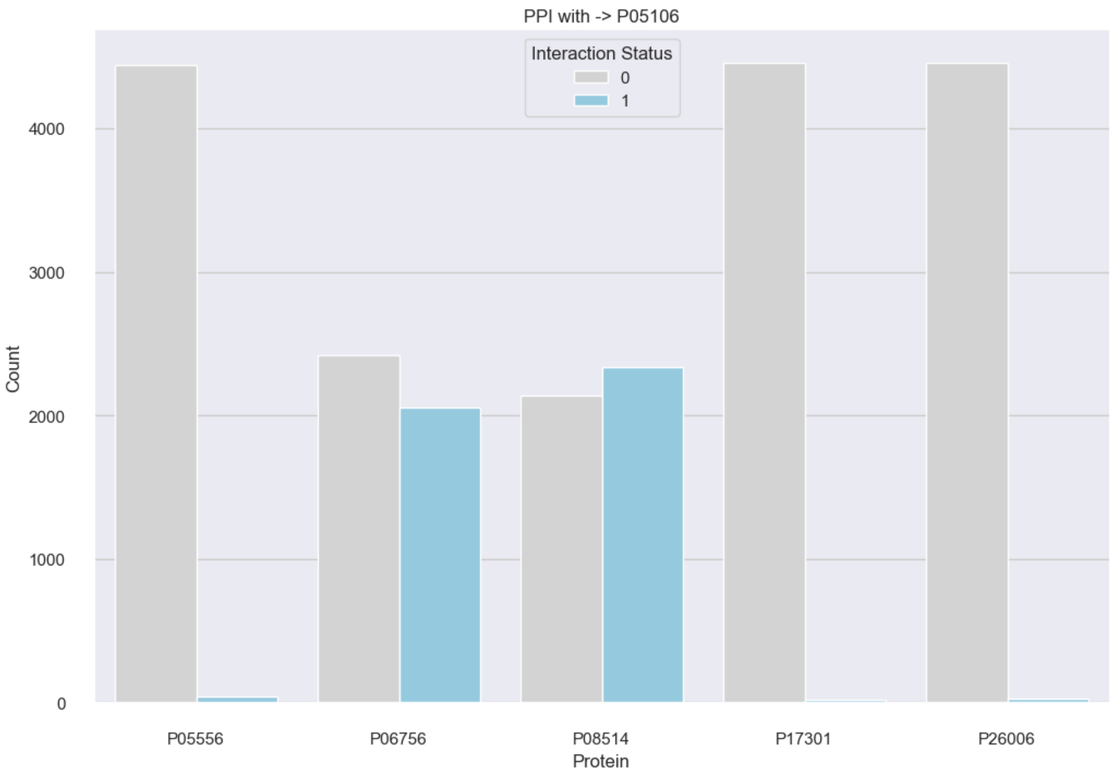
```
[262]: filtered_columns = filter_proteins_list(third_df_encoded, columns_to_remove =␣
         ↪['SMILES', 'P05106'])
       PRINT(f'Filtered columns -> {filtered_columns}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Filtered columns -> ['P05556', 'P06756', 'P08514', 'P17301', 'P26006']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[269]: temp_df_3 = third_df_encoded[filtered_columns]

       temp_df_3.head(5)
```

```
[269]:    P05556  P06756  P08514  P17301  P26006
       0       0       0       0       0       1
       1       0       0       0       0       1
       2       0       0       0       0       1
       3       0       0       0       0       1
       4       0       0       0       0       1
```

```
[270]: visualize_dist(temp_df_3, unique_proteins[2])
```



Explore the Third Data Frame

```
[272]: PRINT(f'The size of the data frame is -> {len(third_df)}')
       print(f'Number of time P17301 appears -> {len(third_df[third_df["UniProt2"] ==␣
         ↪"P17301"])}')
       print(f'Number of time P05556 appears -> {len(third_df[third_df["UniProt2"] ==␣
         ↪"P05556"])}')
       print(f'Number of time P26006 appears -> {len(third_df[third_df["UniProt2"] ==␣
         ↪"P26006"])}')
       print(f'Number of time P06756 appears -> {len(third_df[third_df["UniProt2"] ==␣
         ↪"P06756"])}')
       print(f'Number of time P08514 appears -> {len(third_df[third_df["UniProt2"] ==␣
         ↪"P08514"])}')

       PRINT('Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The size of the data frame is -> 4478
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Number of time P17301 appears -> 20
Number of time P05556 appears -> 37
Number of time P26006 appears -> 25
Number of time P06756 appears -> 2058
```

```
Number of time P08514 appears -> 2338
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Fourth Data Frame

```
[282]: fourth_df = protein_dataframes[unique_proteins[3]]

       fourth_df.head(2)
```

```
[282]:                                               SMILES UniProt1 UniProt2
       0  COC(=O)CN1C(=O)S/C(=C\c2ccc(-c3ccc(C(=O)O)cc3)…   P05107   P11215
       1                   Cc1ccc(/C=C2\SC(=O)N(C)C2=O)o1   P05107   P11215
```

Visualize Distribution

```
[283]: fourth_df_encoded = one_hot_encoding(fourth_df)
```

```
[284]: fourth_df_encoded.columns
```

```
[284]: Index(['SMILES', 'P05107', 'P11215', 'P20701'], dtype='object')
```

```
[285]: fourth_df_encoded.head(3)
```

```
[285]:                                               SMILES  P05107  P11215  P20701
       0  COC(=O)CN1C(=O)S/C(=C\c2ccc(-c3ccc(C(=O)O)cc3)…       1       1       0
       1                   Cc1ccc(/C=C2\SC(=O)N(C)C2=O)o1       1       1       0
       2    CCN1/C(=C/C=C/c2sc3ccccc3[n+]2CC)Sc2ccccc21.[I-]       1       1       0
```

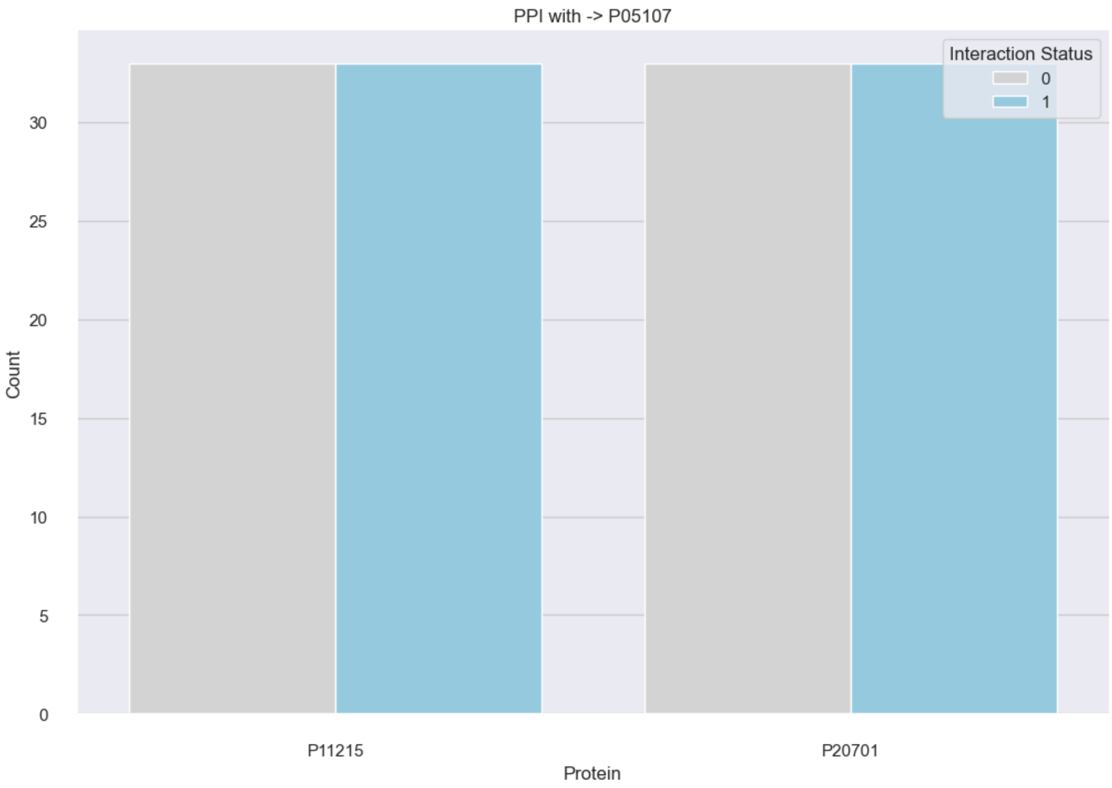
```
[286]: filtered_columns = filter_proteins_list(fourth_df_encoded,␣
         ↪columns_to_remove=['SMILES', 'P05107'])
       PRINT(f'Filtered columns -> {filtered_columns}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Filtered columns -> ['P11215', 'P20701']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[290]: temp_df_4 = fourth_df_encoded[filtered_columns]

       temp_df_4.head(2)
```

```
[290]:    P11215  P20701
       0       1       0
       1       1       0
```

```
[291]: visualize_dist(temp_df_4, unique_proteins[3])
```



The data above is quite interesting, indicating that both proteins appear the same number of times in the PPI with P05107.

Explore the Fourth Data Frame

[294]:
```
PRINT(f'The size of the data frame is -> {len(fourth_df)}')
print(f'Number of time P11215 appears -> {len(fourth_df[fourth_df["UniProt2"]␣
  ↪== "P11215"])}')
print(f'Number of time P20701 appears -> {len(fourth_df[fourth_df["UniProt2"]␣
  ↪== "P20701"])}')

PRINT('Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The size of the data frame is -> 66
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Number of time P11215 appears -> 33
Number of time P20701 appears -> 33
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Fifth Data Frame

[296]:
```
fifth_df = protein_dataframes[unique_proteins[4]]

fifth_df.head(2)
```

[296]:
```
                                              SMILES UniProt1 UniProt2
0  O=C(N[C@@H](Cc1cccc(OCCCCNc2ccccn2)c1)C(=O)O)c…   P08648   P05556
1  CC(C)[C@@H]1NC(=O)[C@@H](Cc2c[nH]c3c(-c4ccc(C(…   P08648   P05556
```

Visualize Distribution

[297]:
```
fifth_df_encoded = one_hot_encoding(fifth_df)
```

[298]:
```
fifth_df_encoded.columns
```

[298]:
```
Index(['SMILES', 'P08648', 'P05556', 'P06756'], dtype='object')
```

[302]:
```
fifth_df_encoded.head(3)
```

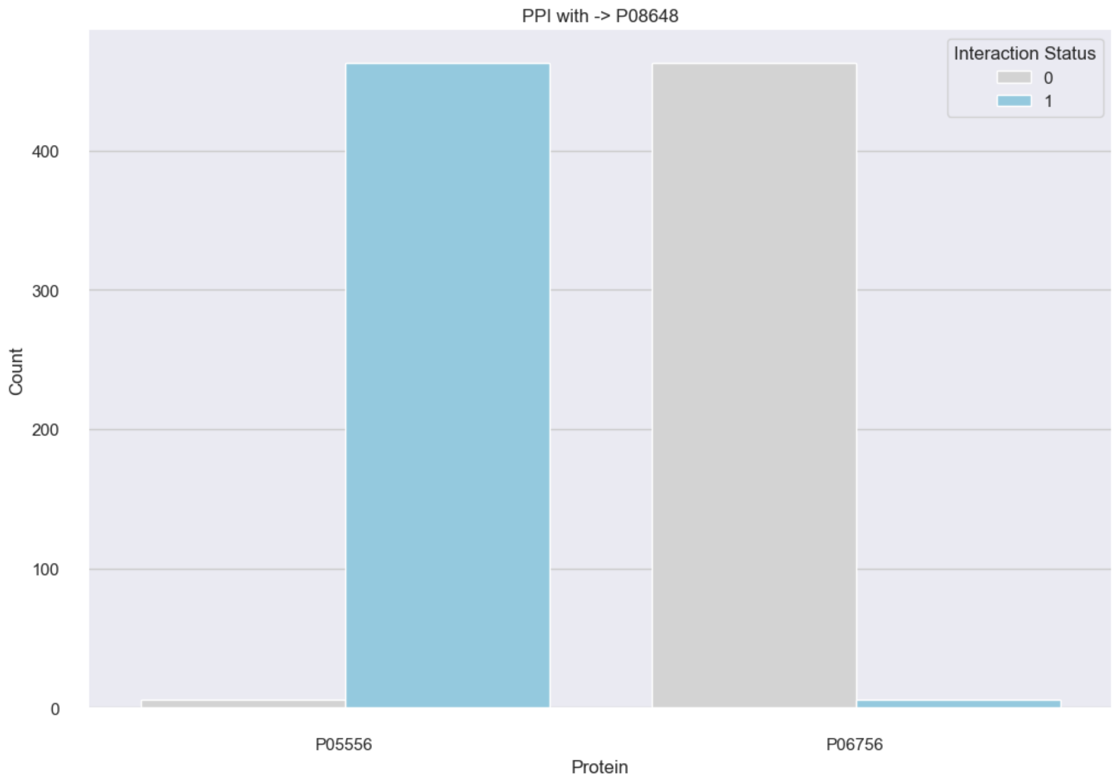
[302]:
```
                                              SMILES  P08648  P05556  P06756
0  O=C(N[C@@H](Cc1cccc(OCCCCNc2ccccn2)c1)C(=O)O)c…       1       1       0
1  CC(C)[C@@H]1NC(=O)[C@@H](Cc2c[nH]c3c(-c4ccc(C(…       1       1       0
2  CC(C)[C@@H]1NC(=O)[C@@H](Cc2c[nH]c3c(-c4ccc5cc…       1       1       0
```

[303]:
```
filtered_columns = filter_proteins_list(fifth_df_encoded,␣
 ↪columns_to_remove=['SMILES', 'P08648'])
PRINT(f'Filtered columns -> {filtered_columns}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Filtered columns -> ['P05556', 'P06756']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[305]:
```
temp_df_5 = fifth_df_encoded[filtered_columns]
```

[307]:
```
visualize_dist(temp_df_5, unique_proteins[4])
```



Here, we observe a particularly interesting distribution of the data. The majority of the fifth dataset represents PPI between the target protein with `UniProt = P08648` and `P05556`. Conversely, there are very few interactions involving `P06756`.

Explore the Fifth Data Frame

[309]:
```
PRINT(f'The size of the data frame is -> {len(fifth_df)}')
print(f'Number of time P05556 appears -> {len(fifth_df[fifth_df["UniProt2"] ==␣
  ↪"P05556"])}')
print(f'Number of time P0756 appears -> {len(fifth_df[fifth_df["UniProt2"] ==␣
  ↪"P06756"])}')

PRINT('Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The size of the data frame is -> 469
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Number of time P05556 appears -> 463
Number of time P0756 appears -> 6
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Sixth Data Frame

```
[310]: sixth_df = protein_dataframes[unique_proteins[5]]

       sixth_df.head(2)
```

```
[310]:                                               SMILES UniProt1 UniProt2
       0  Cc1cccc(Cl)c1C(=O)N[C@@H](Cc1ccc(NC(=O)c2c(Cl)…   P17301   P05556
       1  COc1ccc(S(=O)(=O)N2Cc3[nH]c4ccccc4c3CC2C(N)=O)cc1   P17301   P05556
```

Visualize Distribution

```
[311]: sixth_df_encoded = one_hot_encoding(sixth_df)
```

```
[312]: sixth_df_encoded.columns
```

```
[312]: Index(['SMILES', 'P17301', 'P05106', 'P05556'], dtype='object')
```

```
[313]: sixth_df_encoded.head(3)
```

```
[313]:                                               SMILES  P17301  P05106  P05556
       0  Cc1cccc(Cl)c1C(=O)N[C@@H](Cc1ccc(NC(=O)c2c(Cl)…       1       0       1
       1  COc1ccc(S(=O)(=O)N2Cc3[nH]c4ccccc4c3CC2C(N)=O)cc1       1       0       1
       2  O=C(NCc1ccccc1)NC[C@H](NC(=O)[C@@H]1CCCN1S(=O)…       1       0       1
```

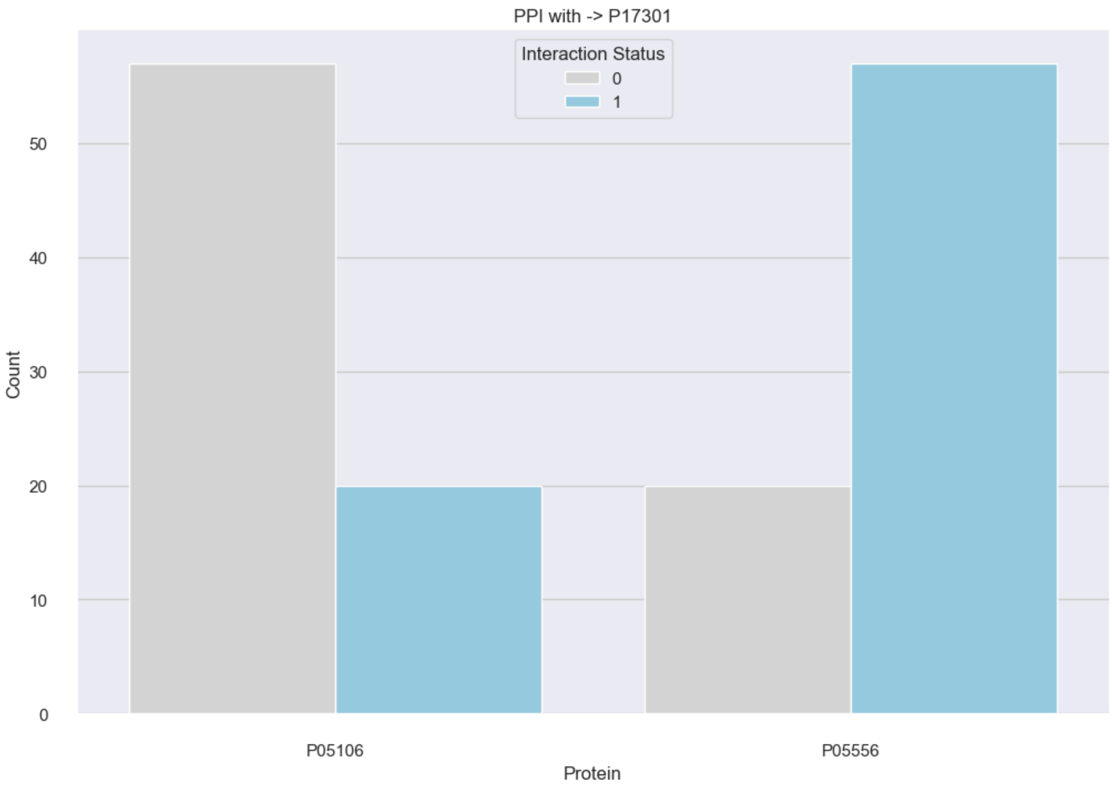
```
[314]: filtered_columns = filter_proteins_list(sixth_df_encoded,␣
        ↪columns_to_remove=['SMILES', 'P17301'])
       PRINT(f'Filtered columns -> {filtered_columns}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Filtered columns -> ['P05106', 'P05556']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[315]: temp_df_6 = sixth_df_encoded[filtered_columns]

       temp_df_6.head(2)
```

```
[315]:    P05106  P05556
       0       0       1
       1       0       1
```

```
[316]: visualize_dist(temp_df_6, unique_proteins[5])
```



Explore the Sixth Dath Frame

[317]:

```
PRINT(f'The size of the data frame is -> {len(sixth_df)}')
print(f'Number of time P05106 appears -> {len(sixth_df[sixth_df["UniProt2"] ==␣
  ↪"P05106"])}')
print(f'Number of time P05556 appears -> {len(sixth_df[sixth_df["UniProt2"] ==␣
  ↪"P05556"])}')

PRINT('Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The size of the data frame is -> 77
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Number of time P05106 appears -> 20
Number of time P05556 appears -> 57
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.2.4 Save the Encoded csv Files

[355]:
```
encoded_dir_path = 'one hot encoded csv files for training'

first_df_encoded.to_csv(os.path.join('one hot encoded csv files for training/
  ↪first_df_encoded.csv'), index=False)
second_df_encoded.to_csv(os.path.join('one hot encoded csv files for training/
  ↪second_df_encoded.csv'), index=False)
third_df_encoded.to_csv(os.path.join('one hot encoded csv files for training/
  ↪third_df_encoded.csv'), index=False)
fourth_df_encoded.to_csv(os.path.join('one hot encoded csv files for training/
  ↪fourth_df_encoded.csv'), index=False)
fifth_df_encoded.to_csv(os.path.join('one hot encoded csv files for training/
  ↪fifth_df_encoded.csv'), index=False)
sixth_df_encoded.to_csv(os.path.join('one hot encoded csv files for training/
  ↪sixth_df_encoded.csv'), index=False)

PRINT('Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.3 Build Classification Models for PPI Prediction

In the upcoming phase, we plan to develop three multiclass classification models to predict Protein-Protein Interactions (PPI) across our six datasets. This entails creating and evaluating six distinct models, one for each dataset. Subsequently, we aim to employ these trained models for predicting PPI on new, unlabeled data.

The models we intend to construct include:

1. Graph Convolution Model using the DeepChem library.
2. Random Forest Multiclass Classifier using the sklearn library.
3. XGBoost Multiclass Classifier.

1.3.1 Graph Convolution Model

Hyperparameter Tuning for the Model

[656]:
```
def gc_model_builder(**model_params):
    """
    Helper function that constructs and configures a GraphConvModel for the PPI␣
  ↪prediction task.
    This function is intended to be used to provide the necessary model for␣
  ↪hyperparameter tuning
    with the `GridHyperparamOpt()` object.

    Parameters:
```

```
    - learning_rate (float): The learning rate for the optimizer.
    - dropout (float): Dropout rate to prevent overfitting.
    - batch_normalize (bool): Whether to apply batch normalization.
    - n_classes (int): Number of classes for classification.

    Returns:
    - GraphConvModel: Configured instance of GraphConvModel for PPI prediction␣
↪tas.
    """
    n_classes = 5 # NEED TO SPECIFY !
    learning_rate = model_params['learning_rate']
    dropout = model_params['dropout']
    batch_normalize = model_params['batch_normalize']

    return GraphConvModel(n_tasks=1,
                          dropout=dropout,
                          mode='classification',
                          batch_normalize=batch_normalize,
                          n_classes=n_classes,
                          learning_rate=learning_rate
                          )
```

[608]:
```
def custom_roc_auc_score(y_true, y_pred, multi_class='ovr', average='weighted'):
    return roc_auc_score(y_true, y_pred, multi_class=multi_class,␣
↪average=average)
```

[855]:
```
def execute_hyperparameter_tuning_for_graph_conv(csv_data, df, params):
    """
    Perform hyperparameter tuning for a Graph Convolutional Model using a grid␣
↪search approach.

    Parameters:
    - csv_data (str or pd.DataFrame): Path to a CSV file containing molecular␣
↪data for `deepchem.data.CSVLoader.featurize()` object
    - df (pd.DataFrame): A Pandas DataFrame containing the data set for the␣
↪model.
    - params (dict): Dictionary of hyperparameters to be tuned.

    Returns:
    - list: A list containing the best hyperparameters and detailed results of␣
↪the hyperparameter search.
    """

    tasks = ['NumericUniProtTargetLabels']
    featurizer = dc.feat.ConvMolFeaturizer()
    loader = dc.data.CSVLoader(tasks=tasks,
                               smiles_field='SMILES',
                               featurizer=featurizer)

    #splitter = dc.splits.RandomSplitter()
    splitter = dc.splits.RandomStratifiedSplitter()

    mean_roc_auc_metric = dc.metrics.Metric(
        metric=custom_roc_auc_score,
        task_averager=np.mean,
        mode='classification',
        n_tasks=1
    )
    dataset = loader.featurize(csv_data)

    res = splitter.train_test_split(dataset, train_frac=0.7)
    train_dataset, valid_dataset = res

    # Create a hyperparameter optimization object
    opt = GridHyperparamOpt(gc_model_builder)
    _, best_hyperparams, all_results = opt.hyperparam_search(params,␣
↪train_dataset, valid_dataset, mean_roc_auc_metric)

    return [best_hyperparams, all_results]
```

[10]:

```
def generate_graph_conv_model(dropout, batch_normalize, n_classes,␣
↪learning_rate, model_dir):
    """
    Generate a Graph Convolutional Neural Network (GraphConvModel) for␣
↪classification tasks.

    Parameters:
    - dropout (float): Dropout rate to apply in the model.
    - batch_normalize (bool): Whether to apply batch normalization.
    - n_classes (int): Number of classes for classification.
    - learning_rate (float): Learning rate for model training.
    - model_dir (str): Directory to save the trained model.

    Returns:
    - model (GraphConvModel): The configured GraphConvModel for classification.
    """
    batch_size = 64
    model = GraphConvModel(n_tasks=1,
                           dropout=dropout,
                           batch_size=batch_size,
                           batch_normalize=batch_normalize,
                           mode='classification',
                           model_dir=model_dir,
                           n_classes=n_classes,
                            learning_rate=learning_rate)

    return model
```

[11]:
```
def GenerateBoxplotForModelPreformaceVisualization(UniProt, cv_folds,␣
 ↪training_score_list, validation_score_list) ->None :
    """
    Generate a boxplot to visualize the performance of a model on training and␣
 ↪validation sets.

    Parameters:
    - UniProt (str): The UniProt ID.
    - cv_folds (int): Number of cross-validation folds.
    - training_score_list (list): List of training scores for each fold.
    - validation_score_list (list): List of validation scores for each fold.

    Returns:
    - None
    """

    data = {
        'Group': ["Training"] * cv_folds + ["Validation"] * cv_folds,
        'Score': training_score_list + validation_score_list
        }

    sns.boxplot(x="Group", y="Score", data=data)

    plt.title(label=f"{UniProt} Mean-Roc-Auc-Score Boxplot Graph",
              fontsize=15,
              color="blue")

    plt.show()
```

[12]:
```
def get_class_labels(predicted_probs):
    """
    Extract class labels from predicted probabilities.

    Parameters:
    - predicted_probs (numpy.ndarray): Array containing predicted probabilities␣
 ↪for each class.

    Returns:
    - class_labels (numpy.ndarray): Array containing the class labels␣
 ↪corresponding to the highest probability.
    """
    # Remove the extra dimension
    squeezed_probs = np.squeeze(predicted_probs, axis=1)
    # Get the class labels
    class_labels = np.argmax(squeezed_probs, axis=1)
    return class_labels
```

1.3.2 Random Forest & XGBoost Multiclass Classifiers Models

Next, our objective is to construct Random Forest and XGBoost multiclass classification models. However, before delving into model development, we recognize the need for additional features to enhance performance. The Graph Convolutional Model from the DeepChem library, employed in our previous model, automatically generates features from the molecular SMILES values during training. Consequently, we were able to feed only two columns, namely SMILES and NumericUniProtTargetLabels, to the model.

In contrast, Random Forest and XGBoost do not generate features during training. To address this, we will utilize RDKitDescriptors & Morgan Fingerprints to generate additional features. These additional features will provide valuable information for our models to learn from, potentially improving their overall performance

Generate Fetures using RDKitDescriptors ####e.

[14]:
```
def calculate_descriptors(smiles):
    """
    Helper function that takes a molecule's SMILES value and generates a list␣
 ↪of the best 8 features
    found to be the most significant for our PPI prediction task.

    Params:
    - smiles (str): Molecule's SMILES value as a string.

    Returns:
    - list: A list of 8 features generated from the molecule's SMILES.
    """
    mol = Chem.MolFromSmiles(smiles)
    if mol is not None:
        descriptors = [
            Descriptors.MolWt(mol),
            Descriptors.NumValenceElectrons(mol),
            Descriptors.TPSA(mol),
            Descriptors.MolLogP(mol),
            Descriptors.NumHeteroatoms(mol),
            Descriptors.NumRotatableBonds(mol),
            Descriptors.HeavyAtomCount(mol),
            Descriptors.FractionCSP3(mol)
        ]
        return descriptors
    else:
        return [None] * 8  # Return None for each descriptor if SMILES cannot␣
 ↪be parsed
```

[15]:

```
def GenerateFeaturesByMoleculeSMILES(df) -> pd.DataFrame:
    """
    Takes a DataFrame containing data for a PPI prediction task and adds␣
  ↪features using the
    `calculate_descriptors(smiles)` feature augmentation helper function.

    Params:
    - df (pd.DataFrame): DataFrame containing data for the task.

    Returns:
    - pd (pd.DataFrame): The same DataFrame after adding the new features.
    """
    df_ = df.copy()
    # Apply the `calculate_descriptors` method in order to generate 8 new␣
  ↪features for df
    df_['MolecularDescriptors'] = df_['SMILES'].apply(calculate_descriptors)

    # Transfer the array at each row under the 'MolecularDescriptors' column␣
  ↪into column with their corresponding names & drop the colunn
    df_[['MolWt', 'NumValenceElectrons', 'TPSA', 'MolLogP', 'NumHeteroatoms',␣
  ↪'NumRotatableBonds', 'HeavyAtomCount', 'FractionCSP3']] = pd.
  ↪DataFrame(df_['MolecularDescriptors'].tolist(), index=df_.index)
    df_.drop(columns=['MolecularDescriptors'], axis=1, inplace=True)

    # Reorder the columns names so that the label column will be the last␣
  ↪column in df
    df_ = df_[['SMILES', 'MolWt', 'NumValenceElectrons', 'TPSA', 'MolLogP',␣
  ↪'NumHeteroatoms', 'NumRotatableBonds', 'HeavyAtomCount', 'FractionCSP3',␣
  ↪'NumericUniProtTargetLabels']]

    return df_
```

Generate Features using Morgan Fingerprints

[16]:

```
def GenerateMorganFingerprintsFeaturesByMoleculeSMILES(df, size, radius) -> pd.
  ↪DataFrame:
    """
    Generate Morgan fingerprints features for molecules based on their SMILES␣
  ↪representation.

    Parameters:
    - df (pd.DataFrame): DataFrame containing data for the task.
    - size (int): Size of the circular fingerprint (number of bits).
    - radius (int): Radius parameter for the circular fingerprint.

    Returns:
    - pd.DataFrame: DataFrame with Morgan fingerprints features added.
    """

    # Define the CircularFingerprint featurizer to generate Morgan Fingerprints␣
  ↪features
    featurizer = dc.feat.CircularFingerprint(size=size, radius=radius)

    # Convert SMILES to features using the featurizer
    X = [featurizer.featurize(smiles) for smiles in df['SMILES']]
    X_flat = [x.flatten() for x in X]
    feature_columns = [f'Feature_{i}' for i in range(len(X_flat[0]))]
    df_features = pd.DataFrame(X_flat, columns=feature_columns)

    # Combine the features with the original dataframe
    df_combined = pd.concat([df, df_features], axis=1)

    df_with_morgan_fingerprints_features = df_combined

    return df_with_morgan_fingerprints_features
```

Buildint Random Forest Multiclass Classifier Model

[318]:

```
def GenerateRandomForestModel(df, weight_dict, n_bootstrap_samples=100,␣
  ↪if_binary=True, bootstrap=True):
    """
    Takes data frame with columns ['SMILES', ... molecule fetures ...,␣
  ↪'NumericUniProtTargetLabels'], traing and evaluate Random Forest Classifier
    model after choosing the best hyperparameters by `GrudSearchCV`. The␣
  ↪function also takes `weight_dict`, which is dictionary of weights assigned
    for each class in case of imbalanced data, or 'balanced' if the data is␣
  ↪balanced.

    Params:
    df - data frame
    weight_dict - dictionary of weight, e.g., {0:1, 1:1.8, 2:1, 3:1.3}. In case␣
  ↪the data balanced, pass 'balanced' instead.
    n_bootstrap_samples - the number of bootstrap samples to create for ROC AUC␣
  ↪calculation
    if_binary - Boolean, True if the number of class is 2, else specify False
    bootstrap - Boolean, True if the data isn't unbalanced much, else Flase.

    Return:
    tuple - (best_rf_model, model_preformance_dictionary)
    """

    # Drop SMILES' and labels columns
    X = df.drop(['SMILES', 'NumericUniProtTargetLabels'], axis=1)
    y = df['NumericUniProtTargetLabels']
```

```
   # Split the dataset into training and test sets
   X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪stratify=y, random_state=42)

   # Generate RF model for hyperparameter tuning phase
   rf_model = RandomForestClassifier(class_weight=weight_dict, random_state=42)

   # Use StratifiedKFold for cross-validation
   stratified_kfold = StratifiedKFold(n_splits=5, shuffle=True,␣
↪random_state=42)

   # Define a parameter distribution
   param_grid = {
       'n_estimators': [50, 100, 150, 200],
       'max_depth': [None, 10, 20, 30],
       'min_samples_split': [2, 5, 10, 15],
       'min_samples_leaf': [1, 2, 4, 8]
   }

   # Use a custom scoring functions for GridSearchCV
   scoring = {
       'Accuracy Score': make_scorer(accuracy_score, average='weighted'),
       'Precision Score': make_scorer(precision_score, average='weighted'),
       'Recall Score': make_scorer(recall_score, average='weighted'),
       'F1 Score': make_scorer(f1_score, average='weighted'),
       'Roc Auc Score': make_scorer(roc_auc_score, needs_proba=True,␣
↪average='weighted', multi_class='ovr')
   }

   # Perform GridSearchCV with StratifiedKFold & get the best hyperparameters
   grid_search = GridSearchCV(rf_model, param_grid, cv=stratified_kfold,␣
↪scoring=scoring, refit='Roc Auc Score')
   grid_search.fit(X_train, y_train)

   best_params = grid_search.best_params_

   # Create a Random Forest classifier with the best hyperparameters
   best_rf_model = RandomForestClassifier(class_weight=weight_dict,␣
↪random_state=42, **best_params)

   # Train the model on the entire training set
   best_rf_model.fit(X_train, y_train)

   # Make predictions on the test set in order to evaluate with the rest␣
↪metrices
```

```python
    y_test_pred = best_rf_model.predict(X_test)

    # Evaluate the model on the test set
    accuracy_test = accuracy_score(y_test, y_test_pred)
    precision_test = precision_score(y_test, y_test_pred, average='weighted')
    recall_test = recall_score(y_test, y_test_pred, average='weighted')
    f1_test = f1_score(y_test, y_test_pred, average='weighted')
    conf_matrix_test = confusion_matrix(y_test, y_test_pred)

    # Print classification report
    print('Classification Report:')
    print(classification_report(y_test, y_test_pred))

    if bootstrap:
        # Initialize arrays to store ROC AUC scores from bootstrap samples
        roc_auc_scores = np.zeros(n_bootstrap_samples)

        # Perform bootstrap resampling and calculate ROC AUC scores
        for i in range(n_bootstrap_samples):
            # Sample with replacement from the test set
            X_bootstrap, y_bootstrap = resample(X_test, y_test, stratify=y_test, random_state=i)

            if if_binary:
                y_bootstrap_pred = best_rf_model.predict(X_bootstrap)
            else:
                y_bootstrap_pred = best_rf_model.predict_proba(X_bootstrap)


            # Calculate ROC AUC score
            roc_auc_scores[i] = roc_auc_score(y_bootstrap, y_bootstrap_pred, average='weighted', multi_class='ovr')

        # Calculate the mean and standard deviation of bootstrap ROC AUC scores
        roc_auc_mean = np.mean(roc_auc_scores)
        roc_auc_std = np.std(roc_auc_scores)
        PRINT(f'Mean ROC AUC: {roc_auc_mean:.3f}, Std Dev ROC AUC: {roc_auc_std:.3f}')

        return (best_rf_model, {
            'accuracy': accuracy_test,
            'roc_auc_mean_score': roc_auc_mean,
            'roc_auc_std_score': roc_auc_std,
            'precision': precision_test,
            'recall': recall_test,
            'f1_score': f1_test,
            'confusion_matrix': conf_matrix_test.tolist()
        })

    # In case bootstrap is False
    else:
        if if_binary:
            y_test_pred = best_rf_model.predict(X_test)
        else:
            y_test_pred = best_rf_model.predict_proba(X_test)

        roc_auc_test = roc_auc_score(y_test, y_test_pred, average='weighted',␣
 ↪multi_class='ovr')
        PRINT(f'ROC AUC {roc_auc_test:.3f}')

        return (best_rf_model, {
            'accuracy': accuracy_test,
            'roc_auc_score': roc_auc_test,
            'precision': precision_test,
            'recall': recall_test,
            'f1_score': f1_test,
            'confusion_matrix': conf_matrix_test.tolist()
        })
```

Buildint XGBoost Multiclass Classifier Model

[333]:

```
def GenerateXGBoostModel(df, weight_dict, n_bootstrap_samples=100,␣
 ↪if_binary=True, bootstrap=True):
    """
    Takes data frame with columns ['SMILES', ... molecule fetures ...,␣
 ↪'NumericUniProtTargetLabels'], traing and evaluate XGBoost model after
    choosing the best hyperparameters by `GrudSearchCV`. The function also␣
 ↪takes `weight_dict`, which is dictionary of weights assigned
    for each class in case of imbalanced data, or 'balanced' if the data is␣
 ↪balanced.

    Params:
    df - data frame
    weight_dict - dictionary of weight, e.g., {0:1, 1:1.8, 2:1, 3:1.3}. In case␣
 ↪the data balanced, pass 'balanced' instead.
    if_binary - Boolean, True if the number of class is 2, else specify False
    bootstrap - Boolean, True if the data isn't unbalanced much, else Flase.

    Return:
    tuple - (best_xbg_model, model_preformance_dictionary)
    """
    # Drop 'SMILES' and labels columns
    X = df.drop(['SMILES', 'NumericUniProtTargetLabels'], axis=1)
    y = df['NumericUniProtTargetLabels']
```

```
# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)

# Generate XGB model for hyperparameter tuning phase
xgb_model = xgb.XGBClassifier(objective='multi:softmax', num_class=len(set(y_train)), random_state=42)

# Use StratifiedKFold for cross-validation
stratified_kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [3, 5],
    'learning_rate': [0.01, 0.1],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0],
    'gamma': [0, 0.2],
    'min_child_weight': [1, 5],
    'reg_alpha': [0, 0.5],
    'reg_lambda': [0, 0.5],
}

# Use a custom scoring functions for GridSearchCV
scoring = {
    'Accuracy Score': make_scorer(accuracy_score, average='weighted'),
    'Precision Score': make_scorer(precision_score, average='weighted'),
    'Recall Score': make_scorer(recall_score, average='weighted'),
    'F1 Score': make_scorer(f1_score, average='weighted'),
    'Roc Auc Score': make_scorer(roc_auc_score, needs_proba=True, average='weighted', multi_class='ovr')
}

# Perform GridSearchCV with StratifiedKFold & extract the best hyperparameters
grid_search = GridSearchCV(xgb_model, param_grid, cv=stratified_kfold, scoring=scoring, refit='Roc Auc Score')
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_

# Create an XGBoost classifier with the best hyperparameters
best_xgb_model = xgb.XGBClassifier(objective='multi:softmax',
                                   num_class=len(set(y_train)),
                                       random_state=42, **best_params)

    # Calculate sample weights for each instance based on class weights
    sample_weights = compute_sample_weight(weight_dict, y_train)

    # Train the model on the entire training set with sample weights
    best_xgb_model.fit(X_train, y_train, sample_weight=sample_weights)

   # Make predictions on the test set in order to evaluate with the rest␣
↪metrices
    y_test_pred = best_xgb_model.predict(X_test)

    # Evaluate the model on the test set
    accuracy_test = accuracy_score(y_test, y_test_pred)
    precision_test = precision_score(y_test, y_test_pred, average='weighted')
    recall_test = recall_score(y_test, y_test_pred, average='weighted')
    f1_test = f1_score(y_test, y_test_pred, average='weighted')
    conf_matrix_test = confusion_matrix(y_test, y_test_pred)

    # Print classification report
    print('Classification Report:')
    print(classification_report(y_test, y_test_pred))

    if bootstrap:
        # Initialize arrays to store ROC AUC scores from bootstrap samples
        roc_auc_scores = np.zeros(n_bootstrap_samples)

        # Perform bootstrap resampling and calculate ROC AUC scores
        for i in range(n_bootstrap_samples):
            # Sample with replacement from the test set
            X_bootstrap, y_bootstrap = resample(X_test, y_test,␣
↪stratify=y_test, random_state=i)

            if if_binary:
                y_bootstrap_pred = best_xgb_model.predict(X_bootstrap)
            else:
                y_bootstrap_pred = best_xgb_model.predict_proba(X_bootstrap)


            # Calculate ROC AUC score
            roc_auc_scores[i] = roc_auc_score(y_bootstrap, y_bootstrap_pred,␣
↪average='weighted', multi_class='ovr')

        # Calculate the mean and standard deviation of bootstrap ROC AUC scores
        roc_auc_mean = np.mean(roc_auc_scores)
        roc_auc_std = np.std(roc_auc_scores)
        PRINT(f'Mean ROC AUC: {roc_auc_mean:.3f}, Std Dev ROC AUC: {roc_auc_std:
↪.3f}')

        return (best_xgb_model, {
            'accuracy': accuracy_test,
            'roc_auc_mean_score': roc_auc_mean,
            'roc_auc_std_score': roc_auc_std,
            'precision': precision_test,
            'recall': recall_test,
            'f1_score': f1_test,
            'confusion_matrix': conf_matrix_test.tolist()
        })

    # In case bootstrap is False
    else:
        if if_binary:
            y_test_pred = best_xgb_model.predict(X_test)
        else:
            y_test_pred = best_xgb_model.predict_proba(X_test)

        roc_auc_test = roc_auc_score(y_test, y_test_pred, average='weighted',␣
↪multi_class='ovr')
        PRINT(f'ROC AUC {roc_auc_test:.3f}')

        return (best_xgb_model, {
            'accuracy': accuracy_test,
            'roc_auc_score': roc_auc_test,
            'precision': precision_test,
            'recall': recall_test,
            'f1_score': f1_test,
            'confusion_matrix': conf_matrix_test.tolist()
        })
```

[432]:
```
def plot_model_comparison_countplots(df):
    """
    Plot bar plots for model performance comparison.

    Parameters:
    - df: DataFrame containing model metrics. Columns represent models, and␣
↪rows represent metrics.
    """

    evaluation_metrices = ['Accuracy', 'Roc Auc Mean Score', 'Roc Auc Std␣
↪Score', 'Precision', 'Recall', 'F1_score']

    # Reshape the DataFrame using melt
    melted_df = pd.melt(df.reset_index(), id_vars=['index'], var_name='Model',␣
↪value_name='Score')

    fig, axes = plt.subplots(nrows=2, ncols=len(df.index)//2, figsize=(10, 6))
    fig.suptitle('Model Performance Bar Plots', fontsize=16)

    # Plot bar plots for each metric
    for i, _ in enumerate(df.index):
        row, col = divmod(i, len(df.index)//2)
        ax = axes[row, col]
        sns.barplot(x='Model', y='Score', data=melted_df[melted_df['index'] ==␣
↪i], ax=ax)
        ax.set_title(f'Plot {chr(65 + i)} - {evaluation_metrices[i]}')
        ax.set_xlabel('Model')
        ax.set_ylabel(f'{evaluation_metrices[i]} Scores')
        ax.tick_params(axis='x', labelsize=8)

    plt.tight_layout(rect=[0, 0, 1, 0.96])
    plt.show()
```

1.4 Build PPI Prediction Model for each Dataset

After constructing three models for our Protein-Protein Interaction (PPI) prediction task, including graph convolution, random forest, and XGBoost multiclass classifiers, the next phase involves developing and training a distinct model for each unique dataset extracted at the beginning of the project.

1.4.1 Construct the Datasets

First we need to load our saved data frames dictionary

[15]:
```
dict_path = 'obj/data_frames_dictionary.pkl'
```

[16]:
```
try:
    with open('obj/data_frames_dictionary.pkl', 'rb') as file:
        df_dict = pickle.load(file)
        PRINT(f'Done.')
except Exception as e:
    PRINT(f'Error in loading the saved data frames dicitonary from obj dir')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[17]:
```
prot_ls = list(df_dict.keys())

PRINT(f'Unique proteins -> {prot_ls}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Unique proteins -> ['P13612', 'P05556', 'P05106', 'P05107', 'P08648', 'P17301']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[18]:

```python
csv_dir_path = 'one hot encoded csv files for training'
```

1.4.2 Prepare the Datasets for Model Training

[19]:

```python
def generate_df_for_training_(UniProt_str, csv_file_name, one_hot_encoded_csv):
    """
    Generate and prepare a DataFrame for model training.

    Parameters:
    - df (pd.DataFrame): Original DataFrame containing the dataset for training.
    - UniProt_str (str): String identifier for the specific UniProt.
    - csv_file_name (str): Name of the CSV file containing UniProt-specific␣
  ↪dataset.
    - one_hot_encoded_csv (str): Name of the CSV file containing one-hot␣
  ↪encoded labels.

    Returns:
    - tuple: A tuple containing two DataFrames: one for model training and the␣
  ↪other with UniProt-specific data.
    """

    # Define the directories for CSV files
    csv_dir = 'unique UniProt csv files'
    csv_dir_ohe = 'one hot encoded csv files for training'

    # Read the UniProt-specific CSV file and the one-hot encoded CSV file
    curr_df = pd.read_csv(os.path.join(csv_dir, csv_file_name))
    ohe_df = pd.read_csv(os.path.join(csv_dir_ohe, one_hot_encoded_csv))

    # Drop unnecessary column and rename the target column
    curr_df.drop('UniProt1', axis=1, inplace=True)
    curr_df = curr_df.rename(columns={'UniProt2':'UniProtTargetLabels'})

    # Extract the list of labels from the one-hot encoded DataFrame
    labels = ohe_df.columns[2:].tolist()

    # Print the UniProt model labels
    PRINT(f'{UniProt_str} model labels -> {labels}')

    # Create a mapping of column names to indices for label encoding
    column_name_to_index = {label: i for i, label in enumerate(labels)}

    # Map the 'labels' column in df to column indices
```

```
    curr_df['NumericUniProtTargetLabels'] = curr_df['UniProtTargetLabels'].
↪map(column_name_to_index)

    # Shuffle the rows
    curr_df = curr_df.sample(frac=1, random_state=42).reset_index(drop=True)

    # Create a DataFrame for model training by dropping the original␣
↪'UniProtTargetLabels' column
    df_for_model = curr_df.drop('UniProtTargetLabels', axis=1)

    PRINT(f'Finished generating DataFrames for UniProt -> {UniProt_str}.')

    # Return the tuple of DataFrames & the label mapping dictionary
    return (df_for_model, curr_df, column_name_to_index)
```

1.4.3 Models for P13612 Protein

[324]:

```
P13612_df_for_training, P13612_df_with_uniprots_col, mapped_label_dict_P13612 =␣
↪generate_df_for_training_('P13612', 'P13612.csv', 'first_df_encoded.csv')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
P13612 model labels -> ['P05556', 'P26010']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Finished generating DataFrames for UniProt -> P13612.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[326]:

```
P13612_df_for_training.head(2)
```

[326]:

```
                                              SMILES  \
0  COc1ccccc1-c1ccc(C[C@H](NC(=O)C2(S(=O)(=O)c3cc…
1  C/C=C/[C@H](CC(=O)O)NC(=O)CN(CCC(C)C)C(=O)Cc1c…

   NumericUniProtTargetLabels
0                           0
1                           0
```

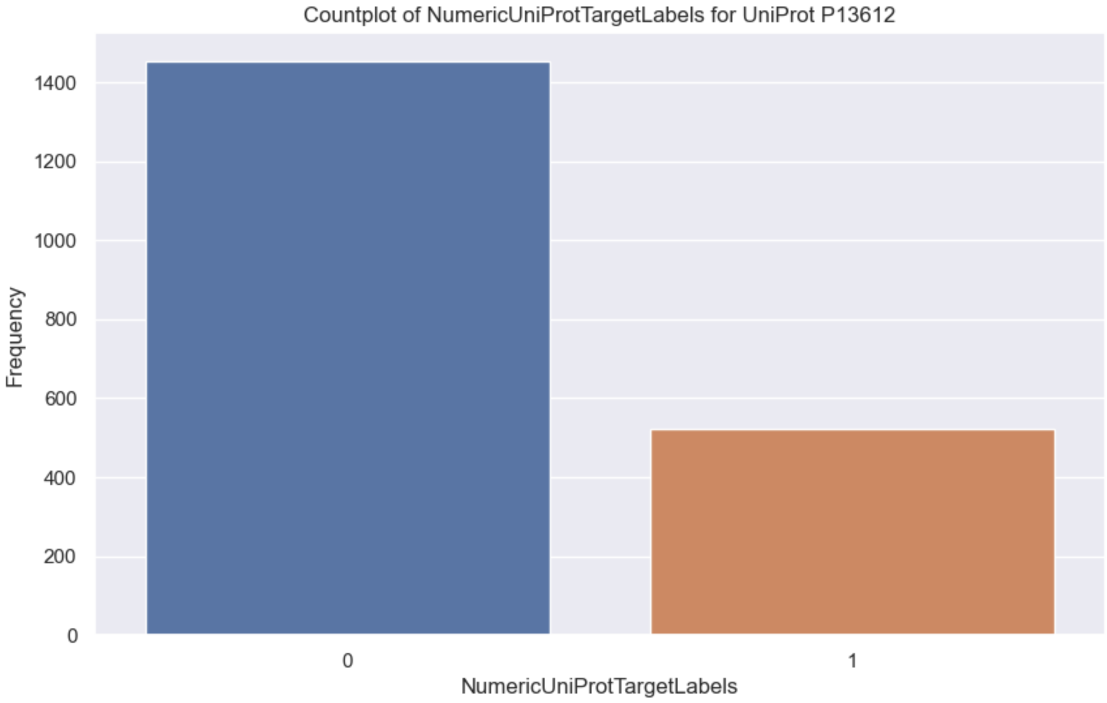
[327]:

```
P13612_df_with_uniprots_col.head(2)
```

[327]:

```
                                              SMILES UniProtTargetLabels  \
0  COc1ccccc1-c1ccc(C[C@H](NC(=O)C2(S(=O)(=O)c3cc…              P05556
1  C/C=C/[C@H](CC(=O)O)NC(=O)CN(CCC(C)C)C(=O)Cc1c…              P05556

   NumericUniProtTargetLabels
0                           0
1                           0
```

```
[328]: plot_uniprot_numeric_label_frequency(df=P13612_df_for_training,␣
         ↪UniProt='P13612')
```



```
[329]: PRINT(f'The mapped labels in ("UniProt": "index_label") format:
         ↪\n\n{mapped_label_dict_P13612}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The mapped labels in ("UniProt": "index_label") format:

{'P05556': 0, 'P26010': 1}
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Quit Dataset Analysis

- **Size of the data frame:** 1974

- **Number of occurrences for each protein:**
  - P05556: 1452
  - P26010: 522

As evident from the data, there is an imbalance in the dataset. To tackle this issue, we plan to assign different weights to the classes. This approach aims to encourage the models to account for the imbalances in the data during training.

Random Forest Multiclass Classifier Model using RKDitDescriptors features for P13612

[330]:

```
P13612_df_for_training_ =␣
  ↪GenerateFeaturesByMoleculeSMILES(df=P13612_df_for_training)
```

[331]:

```
P13612_df_for_training_.head(2)
```

[331]:

```
                                              SMILES    MolWt  \
0  COc1ccccc1-c1ccc(C[C@H](NC(=O)C2(S(=O)(=O)c3cc…  525.598
1  C/C=C/[C@H](CC(=O)O)NC(=O)CN(CCC(C)C)C(=O)Cc1c…  522.646

   NumValenceElectrons    TPSA  MolLogP  NumHeteroatoms  NumRotatableBonds  \
0                  194  109.77  4.39990               9                  9
1                  204  127.84  4.59182               9                 13

   HeavyAtomCount  FractionCSP3  NumericUniProtTargetLabels
0              37      0.285714                           0
1              38      0.379310                           0
```

[332]:

```
weight_dict = 'balanced'

rf_model_tuple_P13612 = GenerateRandomForestModel(df=P13612_df_for_training_,␣
  ↪weight_dict=weight_dict, if_binary=True, bootstrap=True)
PRINT(f'Done training Random Forest Multicalss Classifier Model for UniProt␣
  ↪P13612 using RKDitDescriptors features')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.68      0.75       291
           1       0.40      0.59      0.47       104

    accuracy                           0.66       395
   macro avg       0.61      0.64      0.61       395
weighted avg       0.71      0.66      0.68       395

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 0.635, Std Dev ROC AUC: 0.029
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done training Random Forest Multicalss Classifier Model for UniProt P13612 using
RKDitDescriptors features
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[334]:

```
PRINT(f'The results of Random Forest Multiclass Classifier model␣
  ↪using\nRDKitDescriptors for UniProt P13612 are:')
print_dict_meaningful(rf_model_tuple_P13612[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of Random Forest Multiclass Classifier model using
RDKitDescriptors for UniProt P13612 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.658
roc_auc_mean_score: 0.635
roc_auc_std_score: 0.029
precision: 0.711
recall: 0.658
f1_score: 0.675
confusion_matrix: [[199, 92], [43, 61]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

XGBoost Multiclass Classifier Model using RKDitDescriptors features for P13612

[335]:
```
weight_dict = 'balanced'

xgb_model_tuple_P13612 = GenerateXGBoostModel(df=P13612_df_for_training_,␣
 ↪weight_dict=weight_dict, if_binary=True, bootstrap=True)
PRINT(f'Done training XGBoost Multicalss Classifier Model for UniProt P13612␣
 ↪using RKDitDescriptors features')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.61      0.71       291
           1       0.38      0.67      0.49       104

    accuracy                           0.63       395
   macro avg       0.61      0.64      0.60       395
weighted avg       0.72      0.63      0.65       395

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 0.643, Std Dev ROC AUC: 0.027
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done training XGBoost Multicalss Classifier Model for UniProt P13612 using
RKDitDescriptors features
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[336]:
```
PRINT(f'The results of XGBoost Multiclass Classifier model␣
 ↪using\nRDKitDescriptors features for UniProt P13612 are:')
print_dict_meaningful(xgb_model_tuple_P13612[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of XGBoost Multiclass Classifier model using
```

```
RDKitDescriptors features for UniProt P13612 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.628
roc_auc_mean_score: 0.643
roc_auc_std_score: 0.027
precision: 0.719
recall: 0.628
f1_score: 0.650
confusion_matrix: [[178, 113], [34, 70]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Random Forest Multiclass Classifier Model for P13612 with added Morgan Fingerprints Features

[337]:
```
P13612_df_for_training__ =␣
 ↪GenerateMorganFingerprintsFeaturesByMoleculeSMILES(df=P13612_df_for_training,␣
 ↪size=1024, radius=2)
```

[338]:
```
P13612_df_for_training__.head(2)
```

[338]:
```
                                              SMILES  \
0  COc1ccccc1-c1ccc(C[C@H](NC(=O)C2(S(=O)(=O)c3cc…
1  C/C=C/[C@H](CC(=O)O)NC(=O)CN(CCC(C)C)C(=O)Cc1c…

   NumericUniProtTargetLabels  Feature_0  Feature_1  Feature_2  Feature_3  \
0                           0        0.0        1.0        0.0        0.0
1                           0        0.0        1.0        0.0        0.0

   Feature_4  Feature_5  Feature_6  Feature_7  …  Feature_1014  \
0        1.0        0.0        0.0        0.0  …           0.0
1        0.0        0.0        0.0        0.0  …           0.0

   Feature_1015  Feature_1016  Feature_1017  Feature_1018  Feature_1019  \
0           0.0           0.0           0.0           0.0           1.0
1           0.0           0.0           0.0           0.0           0.0

   Feature_1020  Feature_1021  Feature_1022  Feature_1023
0           0.0           0.0           0.0           0.0
1           0.0           0.0           0.0           0.0

[2 rows x 1026 columns]
```

[339]:
```
weight_dict = 'balanced'

rf_model_tuple_P13612_ = GenerateRandomForestModel(df=P13612_df_for_training__,␣
 ↪weight_dict=weight_dict, if_binary=True, bootstrap=True)
```

```
PRINT(f'Done training Random Forest Multicalss Classifier Model for UniProt␣
  ↪P13612 using Morgan Fingerprints features')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.68      0.81       291
           1       0.52      0.96      0.68       104

    accuracy                           0.76       395
   macro avg       0.75      0.82      0.74       395
weighted avg       0.86      0.76      0.77       395

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 0.824, Std Dev ROC AUC: 0.016
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done training Random Forest Multicalss Classifier Model for UniProt P13612 using
Morgan Fingerprints features
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[342]:

```
PRINT(f'The results of Random Forest Multiclass Classifier model\nusing Morgan␣
  ↪Fingerprints features for UniProt P13612 are:')
print_dict_meaningful(rf_model_tuple_P13612_[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of Random Forest Multiclass Classifier model
using Morgan Fingerprints features for UniProt P13612 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.757
roc_auc_mean_score: 0.824
roc_auc_std_score: 0.016
precision: 0.859
recall: 0.757
f1_score: 0.771
confusion_matrix: [[199, 92], [4, 100]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

XGBoost Multiclass Classifier Model for P13612 with added Morgan Fingerprints Features

[344]:

```
weight_dict = 'balanced'

xgb_model_tuple_P13612_ = GenerateXGBoostModel(df=P13612_df_for_training__,␣
  ↪weight_dict=weight_dict, if_binary=True, bootstrap=True)
```

```
PRINT(f'Done training XGBoost Multicalss Classifier Model for UniProt P13612␣
 ↪using Morgan Fingerprints features')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.61      0.75       291
           1       0.47      0.95      0.63       104

    accuracy                           0.70       395
   macro avg       0.72      0.78      0.69       395
weighted avg       0.84      0.70      0.72       395

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 0.783, Std Dev ROC AUC: 0.015
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done training XGBoost Multicalss Classifier Model for UniProt P13612 using
Morgan Fingerprints features
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[345]:
```
PRINT(f'The results of XGBoost Multiclass Classifier model\nusing Morgan␣
 ↪Fingerprints features for UniProt P13612 are:')
print_dict_meaningful(xgb_model_tuple_P13612_[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of XGBoost Multiclass Classifier model
using Morgan Fingerprints features for UniProt P13612 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.701
roc_auc_mean_score: 0.783
roc_auc_std_score: 0.015
precision: 0.840
recall: 0.701
f1_score: 0.718
confusion_matrix: [[178, 113], [5, 99]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Pickup the best model trained so far for UniProt P13612 As part of each dataset model training pipeline, we will select the best models from our* Random Fores*t and* XGBoos*t multiclass classifier models, utilizing both* RDKit Descripto*rs and* Morgan Fingerprint*s features. This results in a total of four models for each dataset.

Therefore, we will initially visualize the training performances of all four models. We will particularly focus on t*he roc_auc_mean_sc*ore a*nd roc_auc_std_sc*ore metrics when determining the best model. It is important to note that both of these metrics are generated using t*he bootst*rap technique.

Visualize Trained Models Preformances

```
[412]: df_for_preformance_comparison = pd.DataFrame({
           'RM_RDKit': list(rf_model_tuple_P13612[1].values()),
           'XGB_RDKit': list(xgb_model_tuple_P13612[1].values()),
           'RM_MF': list(rf_model_tuple_P13612_[1].values()),
           'XGB_MF': list(xgb_model_tuple_P13612_[1].values())
       }, index=rf_model_tuple_P13612[1].keys())

       df_for_preformance_comparison.head(7)
```

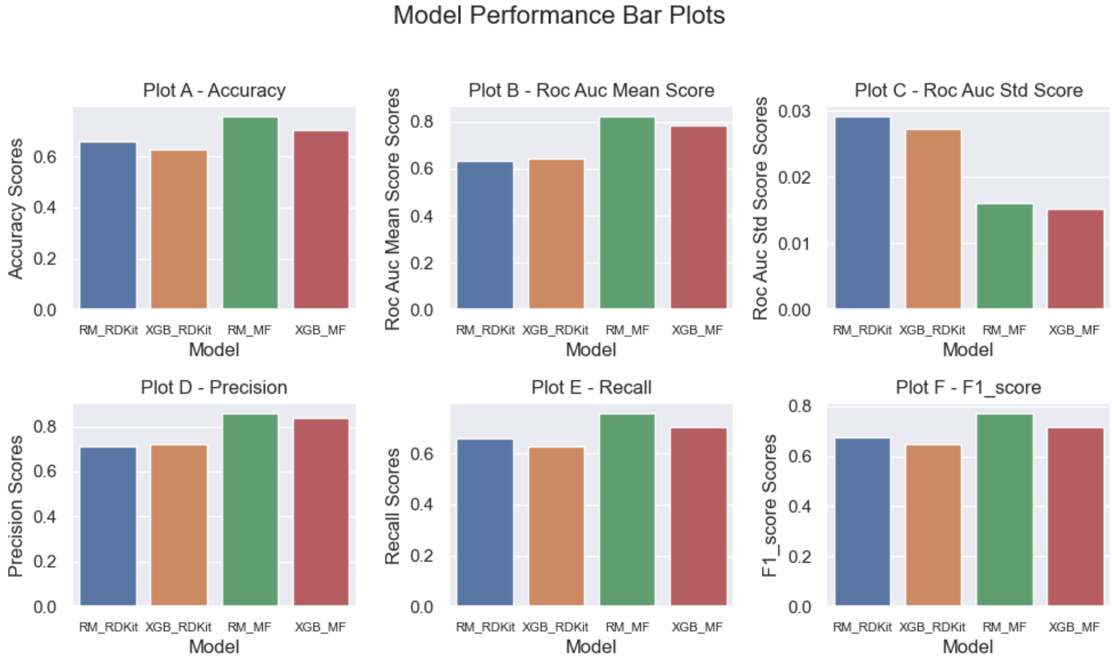
```
[412]:                                  RM_RDKit                XGB_RDKit  \
       accuracy                         0.658228                 0.627848
       roc_auc_mean_score                0.63533                 0.643424
       roc_auc_std_score                0.029222                 0.027219
       precision                        0.710778                  0.71927
       recall                           0.658228                 0.627848
       f1_score                         0.675099                 0.649843
       confusion_matrix     [[199, 92], [43, 61]]  [[178, 113], [34, 70]]

                                           RM_MF                  XGB_MF
       accuracy                         0.756962                0.701266
       roc_auc_mean_score               0.824374                0.783363
       roc_auc_std_score                0.016018                0.015202
       precision                        0.859323                0.839532
       recall                           0.756962                0.701266
       f1_score                         0.771442                0.718282
       confusion_matrix     [[199, 92], [4, 100]]  [[178, 113], [5, 99]]
```

```
[414]: # Drop index column
       df_for_preformance_comparison.drop(df_for_preformance_comparison.index[-1],␣
        ↪inplace=True)

       # Drop the last column (i.e., confusion_metrix column)
       df_for_preformance_comparison.reset_index(drop=True, inplace=True)
```

```
[415]: df_for_preformance_comparison.head(6)
```

```
[415]:    RM_RDKit XGB_RDKit     RM_MF    XGB_MF
       0  0.658228  0.627848  0.756962  0.701266
       1   0.63533  0.643424  0.824374  0.783363
       2  0.029222  0.027219  0.016018  0.015202
       3  0.710778   0.71927  0.859323  0.839532
       4  0.658228  0.627848  0.756962  0.701266
       5  0.675099  0.649843  0.771442  0.718282
```

```
[433]: plot_model_comparison_countplots(df=df_for_preformance_comparison)
```



Choose & Save the Best Model Upon visualizing the plots that represent the performance of all four trained models, it is evident that the Random Forest model utilizing Morgan Fingerprints features achieved the highest roc auc mean score, as well as superior accuracy, precision, recall, and f1 score. Additionally, this model obtained the second-lowest roc auc std score. Consequently, we have decided to select this model.

```
[434]: path = os.path.join('trained models/Best Model of each UniProt', 'rf_P13612_MF.
         ↪joblib')
       dump(rf_model_tuple_P13612_[0], path)

       PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

GraphConvModel Multiclass Classifier Model for P13612

```
[597]: P13612_df_for_training.head(5)
```

```
[597]:                                                  SMILES  \
       0  COc1ccccc1-c1ccc(C[C@H](NC(=O)C2(S(=O)(=O)c3cc…
       1  C/C=C/[C@H](CC(=O)O)NC(=O)CN(CCC(C)C)C(=O)Cc1c…
       2  CC(C)CCNCc1ccccc1-c1ccc(C[C@H](NC(=O)c2ccccc2C…
```

```
3  Cc1c(-c2ccc(C[C@H](NC(=O)c3c(C(C)C)cccc3C(C)C)…
4  CCCCCOc1ccc(C[C@H](NC(=O)[C@@H]2CSCN2C(C)=O)C(…

   NumericUniProtTargetLabels
0                           0
1                           0
2                           0
3                           1
4                           0
```

[598]:

```
csv_dataset_P13612_for_GraphConv_path = os.path.join('data', 'csv Files for␣
 ↪DeepChem GraphConvModel', 'P13612_df_GCM.csv')
```

[ ]:

```
P13612_df_for_training.to_csv(csv_dataset_P13612_for_GraphConv_path,␣
 ↪index=False)
```

Hyperparameter Tuning for Graph Conv Model

[610]:

```
# Define the hyperparameter grid
params = {
    'learning_rate': [1e-3, 5e-4, 1e-4],
    'dropout': [0.2, 0.4],
    'batch_normalize': [True, False],
    'n_classes': [2]
}

# Execute hyperparameter tuning for graph conv model for the first dataset
res_ls = execute_hyperparameter_tuning_for_graph_conv(csv_data =␣
 ↪csv_dataset_P13612_for_GraphConv_path, df=P13612_df_for_training,␣
 ↪params=params)
```

```
smiles_field is deprecated and will be removed in a future version of
DeepChem.Use feature_field instead.
```

[611]:

```
PRINT(f"The results after preforming Grid Hyperparameter Optimization technique␣
 ↪are:")
PRINT(f"Best hyperparameters (learning_rate, dropout, batch_normalize,␣
 ↪n_classes) -> {res_ls[0]}")
PRINT(f"All results :\n\n{list(res_ls[1].values())}")
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results after preforming Grid Hyperparameter Optimization technique are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Best hyperparameters (learning_rate, dropout, batch_normalize, n_classes) ->
(0.001, 0.2, False, 2)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
All results :
```

```
[0.7858676975945017, 0.8157668634162827, 0.6500340882094111, 0.7081515992598465,
0.7250241754417664, 0.7666160571778475, 0.7051777689664287, 0.6746629659000793,
0.6657293036468948, 0.6172515199577056, 0.6635893587903245, 0.5885992715579692]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Build and Train Graph Conv Model

[665]:

```
training_score_list = []
validation_score_list = []
cv_folds = 10

metrics = [dc.metrics.Metric(dc.metrics.roc_auc_score, np.mean,␣
 ↪mode='classification')]

featurizer = dc.feat.ConvMolFeaturizer()
tasks = ['NumericUniProtTargetLabels']
loader = dc.data.CSVLoader(tasks=tasks,
                           smiles_field='SMILES',
                           featurizer=featurizer)
dataset = loader.featurize(csv_dataset_P13612_for_GraphConv_path)
```

```
smiles_field is deprecated and will be removed in a future version of
DeepChem.Use feature_field instead.
```

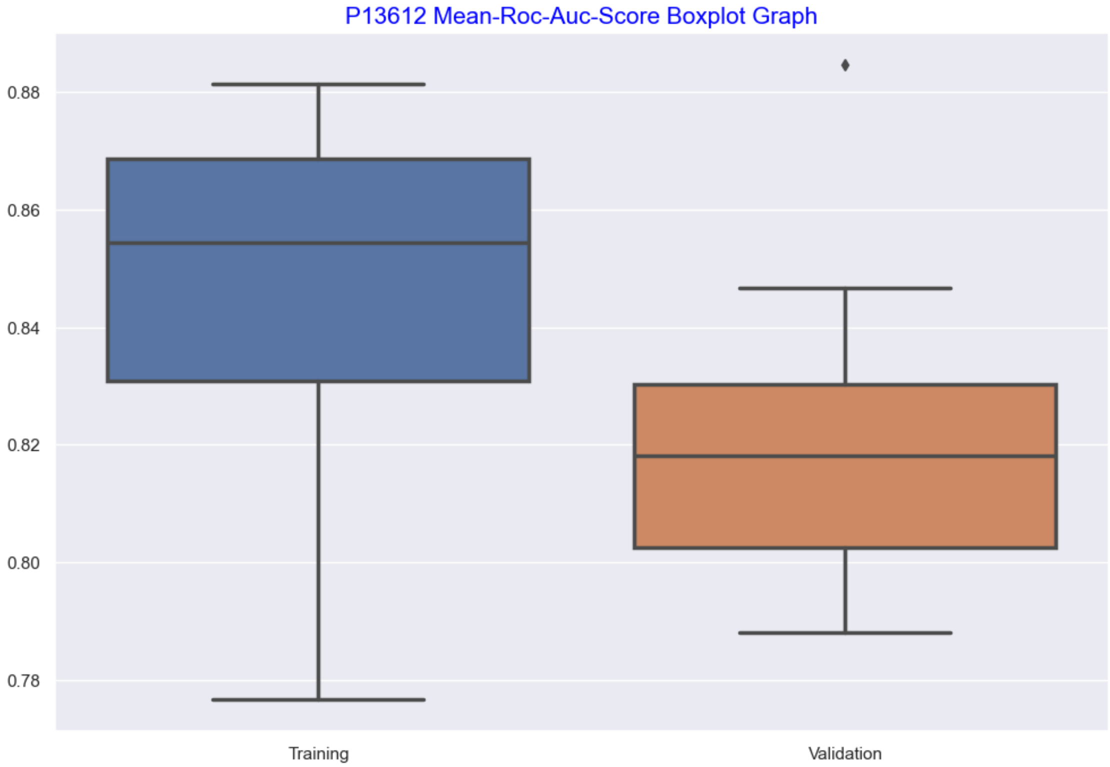
[666]:

```
# Use splitter only once to obtain consistent train/valid splits
splitter = dc.splits.RandomSplitter()

# Create the model outside the loop
model = generate_graph_conv_model(dropout=0.2, batch_normalize=False,␣
 ↪n_classes=2, learning_rate=0.0005, model_dir='models/gcm_P13612')

# Split the data into train&test, save 20% for testing & evaluation of the␣
 ↪trained model
train_dataset, test_dataset = splitter.train_test_split(dataset, test_size=0.2,␣
 ↪random_state=42)

# preforme cs_fold cross validation, in each iteration split the data into␣
 ↪train&val, train the model and test on the validation set.
for i in range(0, cv_folds):
    split = splitter.train_test_split(train_dataset)
    # Split the dataset into train, validation, and test sets
    train_dataset_, valid_dataset_ = split

    # Train the model
    model.fit(train_dataset_, nb_epoch=10)

    # Evaluate on training set
```

```
    train_scores = model.evaluate(train_dataset_, metrics, [], n_classes=2)
    training_score_list.append(train_scores['mean-roc_auc_score'])

    # Evaluate on validation set
    validation_scores = model.evaluate(valid_dataset_, metrics, [], n_classes=2)
    validation_score_list.append(validation_scores['mean-roc_auc_score'])
```

Visualize Model Preformance

[667]:
```
GenerateBoxplotForModelPreformaceVisualization(UniProt='P13612',␣
↪cv_folds=cv_folds , training_score_list=training_score_list ,␣
↪validation_score_list=validation_score_list )
```



Predict on the Test Dataset and Visualize Performance

[668]:
```
# Evaluate on test set
test_scores = model.evaluate(test_dataset, metrics, [], n_classes=2)
test_roc_auc = test_scores['mean-roc_auc_score']

PRINT(f'Mean Roc Auc Score on the test dataset -> ({test_roc_auc:.3f})')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean Roc Auc Score on the test dataset -> (0.826)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Visualize Model Classification Report

[669]:
```
true_labels = test_dataset.y.flatten()
test_predictions = model.predict(test_dataset)


# Get predicted label using helper function
predicted_probs = get_class_labels(predicted_probs=test_predictions)

report = classification_report(true_labels, predicted_probs)

PRINT(report)
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
              precision    recall  f1-score   support

         0.0       0.79      0.89      0.84       288
         1.0       0.56      0.37      0.45       107

    accuracy                           0.75       395
   macro avg       0.67      0.63      0.64       395
weighted avg       0.73      0.75      0.73       395

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

We can observe that we have achieved a fairly good mean ROC AUC score when evaluating our trained *GraphConvModel*, which is quite satisfactory. Additionally, by examining the classification report metric, we can see that our model succeeded in identifying the small class in our unbalanced dataset (label 1).

Therefore, we will consider using this model to make predictions on unseen data for UniProt P13612.

1.4.4 Models for P05556 Protein

Data Cleaning In order to build more generalized and robust models for the dataset of P05556, which found to be extremely unbalanced, with two classes (e.g., O75578 and P23229) with only one sample in the whole dataset !

So, in order to preforme cross validation using stratified_kfold method from sklearn library, we will filter those classes fromk our dataset.

[435]:
```
P05556_df_t = pd.read_csv(os.path.join('unique UniProt csv files', 'P05556.
 ↪csv'))
```

[436]:
```
P05556_df_t.head(5)
```

[436]:
```
                                              SMILES UniProt1 UniProt2
0  CC(C)(C)c1cc(Br)cc([C@H](CC(=O)O)NC(=O)CNC(=O)…   P05556   O75578
1  O=C(NCc1ccccc1)NC[C@H](NC(=O)[C@@H]1CCCN1S(=O)…   P05556   P56199
```

```
2  O=C(NCc1ccccc1)NC[C@H](NC(=O)[C@@H]1CCCN1S(=O)…   P05556   P56199
3  O=C(NCc1ccccc1)NC[C@H](NC(=O)[C@@H]1CCCN1S(=O)…   P05556   P56199
4  Nc1ccc(CNC(=O)NC[C@H](NC(=O)[C@@H]2CCCN2S(=O)(…   P05556   P56199
```

```
[437]: P05556_df_t_ = P05556_df_t[~P05556_df_t['UniProt2'].isin(['O75578', 'P23229'])]
```

```
[438]: PRINT(f'The number of rows we filtered from our dataframe -> {P05556_df_t.
         ↪shape[0] - P05556_df_t_.shape[0]}\nFiltered dataframa shape is ->␣
         ↪{P05556_df_t_.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The number of rows we filtered from our dataframe -> 2
Filtered dataframa shape is -> (2195, 3)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[440]: # Save the filtered data frame as csv file
       P05556_df_t_.to_csv(os.path.join('unique UniProt csv files', 'P05556_.csv'),␣
         ↪index=False)

       PRINT('Saved.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Saved.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[441]: # Generate new one-hot-encoded df of the filtered dataframe & save it
       second_df_encoded_ = one_hot_encoding(P05556_df_t_)

       second_df_encoded_.head(3)
```

```
[441]:                                               SMILES  P05556  P05106  P06756  \
       1  O=C(NCc1ccccc1)NC[C@H](NC(=O)[C@@H]1CCCN1S(=O)…       1       0       0
       2  O=C(NCc1ccccc1)NC[C@H](NC(=O)[C@@H]1CCCN1S(=O)…       1       0       0
       3  O=C(NCc1ccccc1)NC[C@H](NC(=O)[C@@H]1CCCN1S(=O)…       1       0       0

          P08648  P13612  P17301  P56199  Q13797
       1       0       0       0       1       0
       2       0       0       0       1       0
       3       0       0       0       1       0
```

```
[442]: second_df_encoded_.to_csv(os.path.join('one hot encoded csv files for␣
         ↪training', 'second_df_encoded_.csv'), index=False)

       PRINT('Saved.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Saved.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Prepare the Data

[443]:
```
P05556_df_for_training, P05556_df_with_uniprotes_col, mapped_label_dict_P05556␣
↪= generate_df_for_training_('P05556', 'P05556_.csv', 'second_df_encoded_.
↪csv')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
P05556 model labels -> ['P05106', 'P06756', 'P08648', 'P13612', 'P17301',
'P56199', 'Q13797']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Finished generating DataFrames for UniProt -> P05556.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

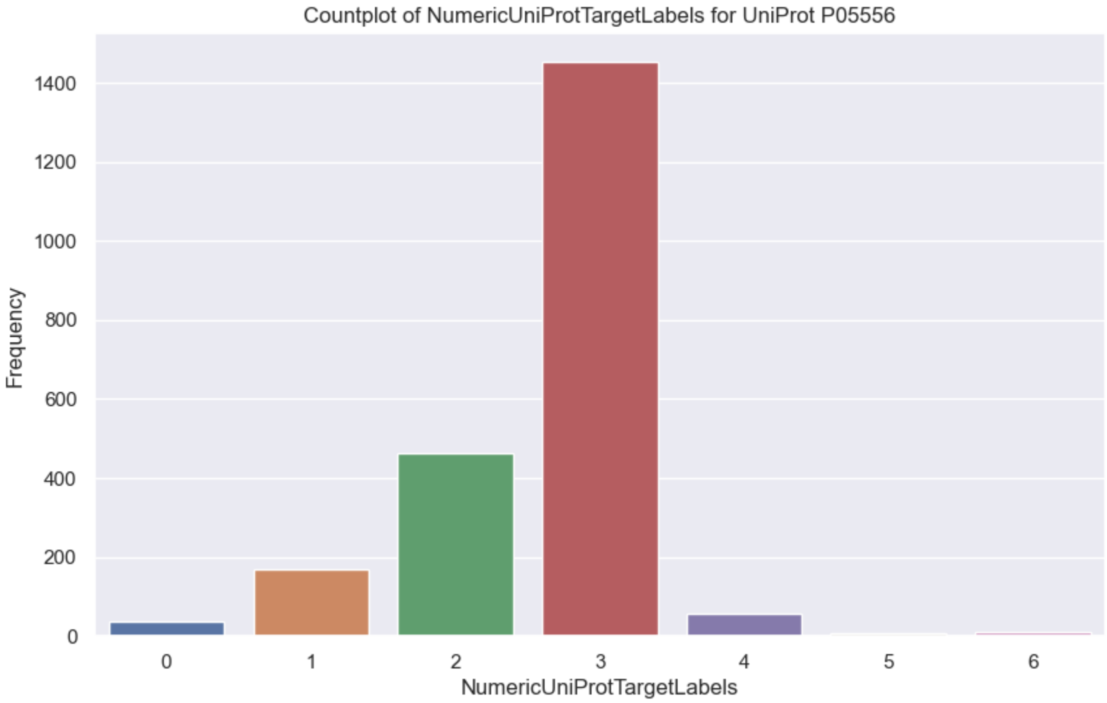
[444]:
```
P05556_df_for_training.head(2)
```

[444]:
```
                                              SMILES  \
0  C=Cc1cnc(NC[C@@H]2C[C@]3(CC(C(=O)NC[C@H](NS(=O…
1  Cl.O=C(O)C1C2C=CC(C2)[C@@H]1NC(=O)[C@@H]1CCCN1…

   NumericUniProtTargetLabels
0                           2
1                           3
```

[445]:
```
P05556_df_with_uniprotes_col.head(2)
```

[445]:
```
                                              SMILES UniProtTargetLabels  \
0  C=Cc1cnc(NC[C@@H]2C[C@]3(CC(C(=O)NC[C@H](NS(=O…              P08648
1  Cl.O=C(O)C1C2C=CC(C2)[C@@H]1NC(=O)[C@@H]1CCCN1…              P13612

   NumericUniProtTargetLabels
0                           2
1                           3
```

[446]:
```
plot_uniprot_numeric_label_frequency(df=P05556_df_for_training,␣
↪UniProt='P05556')
```



[447]:
```
PRINT(f'The mapped labels in ("UniProt": "index_label") format:
↪\n\n{mapped_label_dict_P05556}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The mapped labels in ("UniProt": "index_label") format:

{'P05106': 0, 'P06756': 1, 'P08648': 2, 'P13612': 3, 'P17301': 4, 'P56199': 5,
'Q13797': 6}
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Handle Bad Rows Further in the code, we encountered an exception where the function attempted to extract RDKitDescriptors using a molecule SMILES value of *float* type. We cannot pass this type of value; only *str* type is accepted.

As a solution, we will remove those lines from our dataset.

[448]:
```
PRINT(P05556_df_for_training['SMILES'].apply(type).value_counts())
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
SMILES
<class 'str'>      2194
<class 'float'>       1
Name: count, dtype: int64
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[449]:

```
# Identify rows with 'float' values in the 'SMILES' column
float_rows = P05556_df_for_training['SMILES'].apply(lambda x: isinstance(x,␣
 ↪float))

# Display the rows with 'float' values
float_rows_data = P05556_df_for_training[float_rows]

PRINT(float_rows_data)
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
     SMILES  NumericUniProtTargetLabels
1950    NaN                           2
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[450]:

```
# Drop rows with 'float' values in the 'SMILES' column
P05556_df_for_training = P05556_df_for_training[~float_rows]
```

[451]:

```
PRINT(P05556_df_for_training['SMILES'].apply(type).value_counts())
PRINT('Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
SMILES
<class 'str'>    2194
Name: count, dtype: int64
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Quick Dataset Analyse

- **Size of the data frame:** 2195

- **Number of times each protein appears:**
  - P56199: 6
  - Q13797: 10
  - P17301: 57
  - P05106: 37
  - P06756: 170
  - P08648: 463 - 1 for the *bad row*, thus 462
  - P13612: 1452

Random Forest Multiclass Classifier Model for P05556 with added RDKitDescriptors Features

[452]:

```
P05556_df_for_training_ =␣
 ↪GenerateFeaturesByMoleculeSMILES(df=P05556_df_for_training)
```

```
[454]: P05556_df_for_training_.head(2)
```

```
[454]:
                                               SMILES    MolWt  \
0  C=Cc1cnc(NC[C@@H]2C[C@]3(CC(C(=O)NC[C@H](NS(=O…  659.766
1  Cl.O=C(O)C1C2C=CC(C2)[C@@H]1NC(=O)[C@@H]1CCCN1…  495.812

   NumValenceElectrons    TPSA  MolLogP  NumHeteroatoms  NumRotatableBonds  \
0                  250  204.41  2.46386              16                 14
1                  164  103.78  2.95980              11                  5

   HeavyAtomCount  FractionCSP3  NumericUniProtTargetLabels
0              46      0.500000                           2
1              30      0.473684                           3
```

```
[455]: weight_dict = 'balanced'

       rf_model_tuple_P05556 = GenerateRandomForestModel(df=P05556_df_for_training_,␣
        ↪weight_dict=weight_dict, if_binary=False, bootstrap=True)
       PRINT(f'Done training Random Forest Multicalss Classifier Model for UniProt␣
        ↪P055556 using RKDitDescriptors features')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.23      0.43      0.30         7
           1       0.56      0.71      0.62        34
           2       0.64      0.58      0.61        93
           3       0.91      0.89      0.90       291
           4       0.73      0.73      0.73        11
           5       0.00      0.00      0.00         1
           6       0.00      0.00      0.00         2

    accuracy                           0.79       439
   macro avg       0.44      0.48      0.45       439
weighted avg       0.80      0.79      0.80       439

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 0.924, Std Dev ROC AUC: 0.009
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done training Random Forest Multicalss Classifier Model for UniProt P055556
using RKDitDescriptors features
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[457]: PRINT(f'The results of Random Forest Multiclass Classifier model\nusing␣
        ↪RKDitDescriptors for UniProt P05556 are:')
       print_dict_meaningful(rf_model_tuple_P05556[1])
       PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of Random Forest Multiclass Classifier model
using RKDitDescriptors for UniProt P05556 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.790
roc_auc_mean_score: 0.924
roc_auc_std_score: 0.009
precision: 0.804
recall: 0.790
f1_score: 0.795
confusion_matrix: [[3, 0, 3, 1, 0, 0, 0], [0, 24, 6, 4, 0, 0, 0], [10, 10, 54,
18, 1, 0, 0], [0, 8, 21, 258, 1, 1, 2], [0, 1, 0, 1, 8, 1, 0], [0, 0, 0, 0, 1,
0, 0], [0, 0, 0, 2, 0, 0, 0]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

XGBoost Multiclass Classifier Model for P05556 with added RDKitDescriptors Features

[458]:

```
weight_dict = 'balanced'

xgb_model_tuple_P05556 = GenerateXGBoostModel(df=P05556_df_for_training_,␣
 ↪weight_dict=weight_dict, if_binary=False, bootstrap=True)
PRINT(f'Done training XGBoost Multicalss Classifier Model for UniProt P05556␣
 ↪using RKDitDescriptors features')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.23      0.43      0.30         7
           1       0.49      0.79      0.61        34
           2       0.57      0.61      0.59        93
           3       0.93      0.80      0.86       291
           4       0.64      0.82      0.72        11
           5       0.00      0.00      0.00         1
           6       0.00      0.00      0.00         2

    accuracy                           0.75       439
   macro avg       0.41      0.49      0.44       439
weighted avg       0.79      0.75      0.77       439

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 0.906, Std Dev ROC AUC: 0.010
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done training XGBoost Multicalss Classifier Model for UniProt P05556 using
RKDitDescriptors features
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Random Forest Multiclass Classifier Model for P05556 with added Morgan Fingerprints Features

```
[459]: P05556_df_for_training__ =␣
        ↪GenerateMorganFingerprintsFeaturesByMoleculeSMILES(df=P05556_df_for_training,␣
        ↪size=1024, radius=2)
```

```
[467]: P05556_df_for_training__.head(2)
```

```
[467]:                                                   SMILES  \
       0  C=Cc1cnc(NC[C@@H]2C[C@]3(CC(C(=O)NC[C@H](NS(=O…
       1  Cl.O=C(O)C1C2C=CC(C2)[C@@H]1NC(=O)[C@@H]1CCCN1…

          NumericUniProtTargetLabels  Feature_0  Feature_1  Feature_2  Feature_3  \
       0                         2.0        0.0        1.0        0.0        0.0
       1                         3.0        0.0        0.0        0.0        0.0

          Feature_4  Feature_5  Feature_6  Feature_7  …  Feature_1014  \
       0        0.0        0.0        0.0        0.0  …           0.0
       1        1.0        0.0        0.0        0.0  …           0.0

          Feature_1015  Feature_1016  Feature_1017  Feature_1018  Feature_1019  \
       0           0.0           0.0           0.0           0.0           1.0
       1           0.0           0.0           0.0           0.0           1.0

          Feature_1020  Feature_1021  Feature_1022  Feature_1023
       0           0.0           1.0           0.0           0.0
       1           0.0           0.0           0.0           0.0

       [2 rows x 1026 columns]
```

Check for Generated Features With NaN values

```
[461]: original_rows = P05556_df_for_training__.shape[0]

       P05556_df_for_training__ = P05556_df_for_training__.dropna()

       # Calculate the number of dropped rows
       dropped_rows = original_rows - P05556_df_for_training__.shape[0]

       PRINT(f"{dropped_rows} rows were dropped.")
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
2 rows were dropped.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[462]: weight_dict = 'balanced'

       rf_model_tuple_P05556_ = GenerateRandomForestModel(df=P05556_df_for_training__,␣
        ↪weight_dict=weight_dict, if_binary=False, bootstrap=True)
```

```
PRINT(f'Done training Random Forest Multicalss Classifier Model for UniProt␣
 ↪P055556 using Morgan Fingerprints features')
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.31      0.71      0.43         7
         1.0       0.57      0.71      0.63        34
         2.0       0.69      0.56      0.62        93
         3.0       0.91      0.90      0.90       291
         4.0       0.64      0.64      0.64        11
         5.0       0.00      0.00      0.00         1
         6.0       0.33      0.50      0.40         2

    accuracy                           0.80       439
   macro avg       0.49      0.57      0.52       439
weighted avg       0.81      0.80      0.80       439

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 0.895, Std Dev ROC AUC: 0.013
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done training Random Forest Multicalss Classifier Model for UniProt P055556
using Morgan Fingerprints features
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[464]:

```
PRINT(f'The results of Random Forest Multiclass Classifier using\nMorgan␣
 ↪Fingerprints features model for UniProt P05556 are:')
print_dict_meaningful(rf_model_tuple_P05556_[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of Random Forest Multiclass Classifier using
Morgan Fingerprints features model for UniProt P05556 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.800
roc_auc_mean_score: 0.895
roc_auc_std_score: 0.013
precision: 0.815
recall: 0.800
f1_score: 0.804
confusion_matrix: [[5, 0, 2, 0, 0, 0, 0], [0, 24, 5, 5, 0, 0, 0], [11, 10, 52,
18, 2, 0, 0], [0, 8, 16, 262, 1, 2, 2], [0, 0, 0, 3, 7, 1, 0], [0, 0, 0, 0, 1,
0, 0], [0, 0, 0, 1, 0, 0, 1]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

XGBoost Multiclass Classifier Model for P05556 with added Morgan Fingerprints Features

```
[468]: weight_dict = 'balanced'

       xgb_model_tuple_P05556_ = GenerateXGBoostModel(df=P05556_df_for_training__,␣
        ↪weight_dict=weight_dict, if_binary=False, bootstrap=True)
       PRINT(f'Done training XGBoost Multicalss Classifier Model for UniProt P05556␣
        ↪using Morgan Fingerprints features')
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.29      0.86      0.43         7
         1.0       0.57      0.71      0.63        34
         2.0       0.69      0.55      0.61        93
         3.0       0.91      0.88      0.90       291
         4.0       0.58      0.64      0.61        11
         5.0       0.00      0.00      0.00         1
         6.0       0.20      0.50      0.29         2

    accuracy                           0.79       439
   macro avg       0.46      0.59      0.49       439
weighted avg       0.81      0.79      0.80       439

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 0.885, Std Dev ROC AUC: 0.014
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done training XGBoost Multicalss Classifier Model for UniProt P05556 using
Morgan Fingerprints features
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[469]: PRINT(f'The results of XGBoost Multiclass Classifier using\nMorgan Fingerprints␣
        ↪features model for UniProt P05556 are:')
       print_dict_meaningful(xgb_model_tuple_P05556_[1])
       PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of XGBoost Multiclass Classifier using
Morgan Fingerprints features model for UniProt P05556 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.788
roc_auc_mean_score: 0.885
roc_auc_std_score: 0.014
precision: 0.814
recall: 0.788
f1_score: 0.796
confusion_matrix: [[6, 0, 1, 0, 0, 0, 0], [0, 24, 4, 6, 0, 0, 0], [14, 11, 51,
```

```
15, 2, 0, 0], [1, 7, 18, 257, 2, 2, 4], [0, 0, 0, 3, 7, 1, 0], [0, 0, 0, 0, 1,
0, 0], [0, 0, 0, 1, 0, 0, 1]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Pickup the best model trained so far for UniProt P05556

Visualize Trained Models Preformances

[470]:
```
df_for_preformance_comparison = pd.DataFrame({
    'RM_RDKit': list(rf_model_tuple_P05556[1].values()),
    'XGB_RDKit': list(xgb_model_tuple_P05556[1].values()),
    'RM_MF': list(rf_model_tuple_P05556_[1].values()),
    'XGB_MF': list(xgb_model_tuple_P05556_[1].values())
}, index=rf_model_tuple_P13612[1].keys())

df_for_preformance_comparison.head(7)
```

[470]:
```
                                                            RM_RDKit  \
accuracy                                                    0.790433
roc_auc_mean_score                                          0.924319
roc_auc_std_score                                           0.009041
precision                                                   0.803501
recall                                                      0.790433
f1_score                                                    0.795402
confusion_matrix    [[3, 0, 3, 1, 0, 0, 0], [0, 24, 6, 4, 0, 0, 0]…

                                                           XGB_RDKit  \
accuracy                                                    0.751708
roc_auc_mean_score                                          0.905885
roc_auc_std_score                                           0.010085
precision                                                   0.794082
recall                                                      0.751708
f1_score                                                    0.766261
confusion_matrix    [[3, 0, 4, 0, 0, 0, 0], [0, 27, 5, 2, 0, 0, 0]…

                                                               RM_MF  \
accuracy                                                    0.799544
roc_auc_mean_score                                          0.895264
roc_auc_std_score                                           0.012737
precision                                                   0.814524
recall                                                      0.799544
f1_score                                                    0.803627
confusion_matrix    [[5, 0, 2, 0, 0, 0, 0], [0, 24, 5, 5, 0, 0, 0]…

                                                              XGB_MF
accuracy                                                    0.788155
```

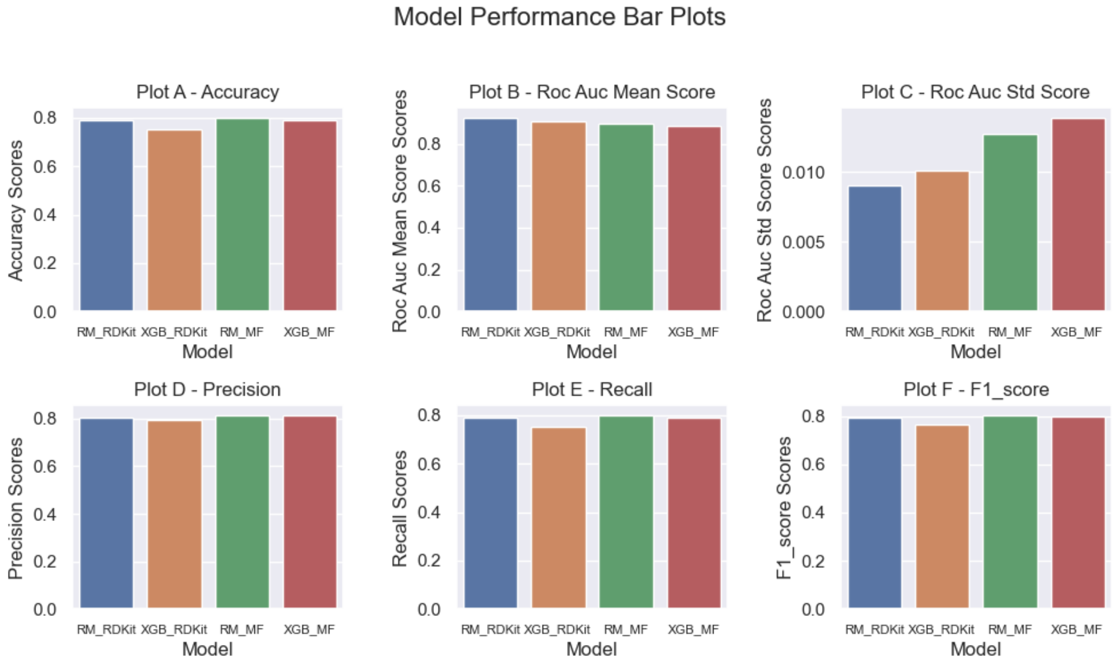
```
roc_auc_mean_score                                            0.88473
roc_auc_std_score                                            0.013897
precision                                                    0.814446
recall                                                       0.788155
f1_score                                                     0.796309
confusion_matrix    [[6, 0, 1, 0, 0, 0, 0], [0, 24, 4, 6, 0, 0, 0]…
```

[471]:
```
# Drop index columm
df_for_preformance_comparison.drop(df_for_preformance_comparison.index[-1],␣
 ↪inplace=True)

# Drop the last row (i.e., confusion_matrix row)
df_for_preformance_comparison.reset_index(drop=True, inplace=True)
```

[472]:
```
df_for_preformance_comparison.head(6)
```

[472]:
```
   RM_RDKit XGB_RDKit     RM_MF    XGB_MF
0  0.790433  0.751708  0.799544  0.788155
1  0.924319  0.905885  0.895264   0.88473
2  0.009041  0.010085  0.012737  0.013897
3  0.803501  0.794082  0.814524  0.814446
4  0.790433  0.751708  0.799544  0.788155
5  0.795402  0.766261  0.803627  0.796309
```

[473]:
```
plot_model_comparison_countplots(df=df_for_preformance_comparison)
```



Choose & Save the Best Model Upon examining the plots illustrating the performance of all four trained models, it is evident that each of them achieved high scores in most evaluation metrics. However, upon closer inspection of plot C, which represents the roc auc std score, we observe that the Random Forest model trained with RDKit descriptors features exhibits the lowest standard deviation in the roc auc score generated using the bootstrap technique.

As a result, we will designate this particular model as the chosen candidate for being the best among the four models for UniProt P05556.

[474]:
```
path = os.path.join('trained models/Best Model of each UniProt',␣
 ↪'rf_P05556_RDKD.joblib')
dump(rf_model_tuple_P05556[0], path)

PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.4.5 Models for P05106 Protein

[478]:
```
P05106_df_for_training, P05106_df_with_uniprotes_col, mapped_label_dict_P05106␣
 ↪= generate_df_for_training_('P05106', 'P05106.csv', 'third_df_encoded.csv')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
P05106 model labels -> ['P05556', 'P06756', 'P08514', 'P17301', 'P26006']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Finished generating DataFrames for UniProt -> P05106.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[479]:
```
P05106_df_for_training.head(3)
```

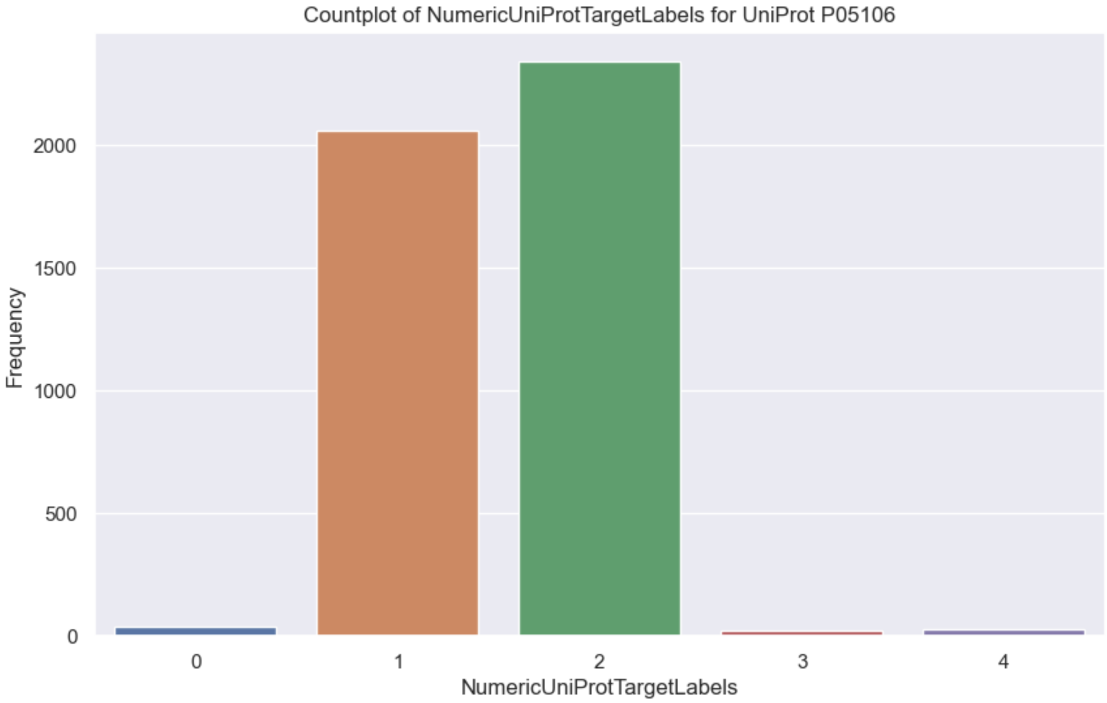
[479]:
```
                                              SMILES  \
0  COC(=O)c1ccc(COC(=O)N[C@@H](CCC(=O)N2CCN(c3ccc…
1  COCCOCCOCCOCC(=O)Nc1cc(C[C@H](NS(=O)(=O)c2cccc…
2  Cl.O=C(NC(Cc1ccc2cc(OCCCN3CCNCC3)ccc2c1)C(=O)O…

   NumericUniProtTargetLabels
0                           1
1                           1
2                           2
```

[480]:
```
P05106_df_with_uniprotes_col.head(3)
```

[480]:
```
                                              SMILES UniProtTargetLabels  \
0  COC(=O)c1ccc(COC(=O)N[C@@H](CCC(=O)N2CCN(c3ccc…              P06756
1  COCCOCCOCCOCC(=O)Nc1cc(C[C@H](NS(=O)(=O)c2cccc…              P06756
```

```
2  Cl.O=C(NC(Cc1ccc2cc(OCCCN3CCNCC3)ccc2c1)C(=O)O…            P08514

   NumericUniProtTargetLabels
0                           1
1                           1
2                           2
```

[481]:
```
plot_uniprot_numeric_label_frequency(df=P05106_df_for_training,␣
  ↪UniProt='P05106')
```



[482]:
```
PRINT(f'The mapped labels in ("UniProt": "index_label") format:
  ↪\n\n{mapped_label_dict_P05106}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The mapped labels in ("UniProt": "index_label") format:

{'P05556': 0, 'P06756': 1, 'P08514': 2, 'P17301': 3, 'P26006': 4}
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Handle Bad Rows Further in the code, we encountered an exception where the function attempted to extract RDKitDescriptors using a molecule SMILES value of *float* type. We cannot pass this type of value; only *str* type is accepted.

As a solution, we will remove those lines from our dataset.

```
[483]: PRINT(P05106_df_for_training['SMILES'].apply(type).value_counts())
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
SMILES
<class 'str'>      4475
<class 'float'>       3
Name: count, dtype: int64
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[484]: # Identify rows with 'float' values in the 'SMILES' column
float_rows = P05106_df_for_training['SMILES'].apply(lambda x: isinstance(x,␣
 ↪float))

# Display the rows with 'float' values
float_rows_data = P05106_df_for_training[float_rows]

PRINT(float_rows_data)
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
     SMILES  NumericUniProtTargetLabels
3434    NaN                           2
4202    NaN                           2
4343    NaN                           1
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[485]: # Drop rows with 'float' values in the 'SMILES' column
P05106_df_for_training = P05106_df_for_training[~float_rows]
```

```
[486]: PRINT(P05106_df_for_training['SMILES'].apply(type).value_counts())
PRINT('Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
SMILES
<class 'str'>    4475
Name: count, dtype: int64
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Quick Dataset Analyse

- **Size of the data frame:** 4478 - 3 =4475

- **Number of times each protein appears:**
  - P17301: 20
  - P05556: 37
  - P26006: 25

  - P06756: 2058 - 1 = 2057
  - P08514: 2338 - 2 = 2336

Random Forest Multiclass Classifier Model for P05106 with added RDKitDescriptors Features

[487]:
```
P05106_df_for_training_ =␣
  ↪GenerateFeaturesByMoleculeSMILES(df=P05106_df_for_training)
```

[488]:
```
P05106_df_for_training_.head(2)
```

[488]:
```
                                              SMILES    MolWt  \
0  COC(=O)c1ccc(COC(=O)N[C@@H](CCC(=O)N2CCN(c3ccc…  580.642
1  COCCOCCOCCOCC(=O)Nc1cc(C[C@H](NS(=O)(=O)c2cccc…  797.850

   NumValenceElectrons    TPSA  MolLogP  NumHeteroatoms  NumRotatableBonds  \
0                  224  161.90   2.0428              13                 10
1                  302  195.67   3.9484              19                 24

   HeavyAtomCount  FractionCSP3  NumericUniProtTargetLabels
0              42      0.413793                           1
1              55      0.472222                           1
```

[489]:
```
weight_dict = 'balanced'

rf_model_tuple_P05106 = GenerateRandomForestModel(df=P05106_df_for_training_,␣
  ↪weight_dict=weight_dict, if_binary=False, bootstrap=True)
PRINT(f'Done training Random Forest Multicalss Classifier Model for UniProt␣
  ↪P05106 using RKDitDescriptors features')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.44      0.50      0.47         8
           1       0.61      0.60      0.60       411
           2       0.67      0.67      0.67       467
           3       0.00      0.00      0.00         4
           4       0.27      0.60      0.37         5

    accuracy                           0.63       895
   macro avg       0.40      0.47      0.42       895
weighted avg       0.63      0.63      0.63       895

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 0.742, Std Dev ROC AUC: 0.016
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done training Random Forest Multicalss Classifier Model for UniProt P05106 using
```

```
RKDitDescriptors features
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[490]:

```
PRINT(f'The results of Random Forest Multiclass Classifier model\nusing RDKitDescriptors features for UniProt P05106 are:')
print_dict_meaningful(rf_model_tuple_P05106[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of Random Forest Multiclass Classifier model
using RDKitDescriptors features for UniProt P05106 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.631
roc_auc_mean_score: 0.742
roc_auc_std_score: 0.016
precision: 0.632
recall: 0.631
f1_score: 0.631
confusion_matrix: [[4, 3, 1, 0, 0], [5, 247, 154, 0, 5], [0, 152, 311, 1, 3],
[0, 3, 1, 0, 0], [0, 1, 0, 1, 3]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

XGBoost Multiclass Classifier Model using RKDitDescriptors features for P05106

[491]:

```
weight_dict = 'balanced'

xgb_model_tuple_P05106 = GenerateXGBoostModel(df=P05106_df_for_training_, weight_dict=weight_dict, if_binary=False, bootstrap=True)
PRINT(f'Done training XGBoost Multicalss Classifier Model for UniProt P05106 using RKDitDescriptors features')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.36      0.62      0.45         8
           1       0.63      0.64      0.64       411
           2       0.69      0.67      0.68       467
           3       0.00      0.00      0.00         4
           4       0.38      0.60      0.46         5

    accuracy                           0.65       895
   macro avg       0.41      0.51      0.45       895
weighted avg       0.66      0.65      0.65       895

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 0.756, Std Dev ROC AUC: 0.015
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done training XGBoost Multicalss Classifier Model for UniProt P05106 using
RKDitDescriptors features
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[492]: PRINT(f'The results of the best XGBoost Multiclass Classifier model␣
         ↪for\nUniProt P05106 are:')
       print_dict_meaningful(xgb_model_tuple_P05106[1])
       PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of the best XGBoost Multiclass Classifier model for
UniProt P05106 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.653
roc_auc_mean_score: 0.756
roc_auc_std_score: 0.015
precision: 0.658
recall: 0.653
f1_score: 0.655
confusion_matrix: [[5, 2, 1, 0, 0], [7, 264, 136, 1, 3], [1, 148, 312, 4, 2],
[1, 2, 1, 0, 0], [0, 1, 0, 1, 3]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Random Forest Multiclass Classifier Model for P05106 with added Morgan Fingerprints Features

```
[507]: P05106_df_for_training__ =␣
         ↪GenerateMorganFingerprintsFeaturesByMoleculeSMILES(df=P05106_df_for_training,␣
         ↪size=1024, radius=2)
```

```
[508]: P05106_df_for_training__.head(2)
```

```
[508]:                                              SMILES  \
        0  COC(=O)c1ccc(COC(=O)N[C@@H](CCC(=O)N2CCN(c3ccc…
        1  COCCOCCOCCOCC(=O)Nc1cc(C[C@H](NS(=O)(=O)c2cccc…

           NumericUniProtTargetLabels  Feature_0  Feature_1  Feature_2  Feature_3  \
        0                         1.0        0.0        1.0        0.0        0.0
        1                         1.0        0.0        1.0        0.0        0.0

           Feature_4  Feature_5  Feature_6  Feature_7  …  Feature_1014  \
        0        1.0        0.0        0.0        0.0  …           0.0
        1        1.0        0.0        0.0        0.0  …           0.0

           Feature_1015  Feature_1016  Feature_1017  Feature_1018  Feature_1019  \
        0           0.0           0.0           0.0           1.0           0.0
```

```
1          0.0          0.0          0.0          0.0          1.0

   Feature_1020  Feature_1021  Feature_1022  Feature_1023
0           0.0           0.0           0.0           0.0
1           0.0           0.0           0.0           0.0

[2 rows x 1026 columns]
```

Handle Bad Rows

[509]:
```
original_rows = P05106_df_for_training__.shape[0]

P05106_df_for_training__ = P05106_df_for_training__.dropna()

# Calculate the number of dropped rows
dropped_rows = original_rows - P05106_df_for_training__.shape[0]

PRINT(f"{dropped_rows} rows were dropped.")
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
6 rows were dropped.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[510]:
```
# Conver target labels into int
P05106_df_for_training__['NumericUniProtTargetLabels'] =␣
 ↪P05106_df_for_training__['NumericUniProtTargetLabels'].astype(int)
```

[511]:
```
weight_dict = 'balanced'

rf_model_tuple_P05106_ = GenerateRandomForestModel(df=P05106_df_for_training__,␣
 ↪weight_dict=weight_dict, if_binary=False, bootstrap=True)
PRINT(f'Done training Random Forest Multicalss Classifier Model for UniProt␣
 ↪P05106 using Morgan Fingerprints features')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.38      0.62      0.48         8
           1       0.63      0.68      0.66       411
           2       0.70      0.64      0.67       467
           3       0.22      0.50      0.31         4
           4       0.67      0.40      0.50         5

    accuracy                           0.66       895
   macro avg       0.52      0.57      0.52       895
weighted avg       0.67      0.66      0.66       895

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 0.749, Std Dev ROC AUC: 0.015
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done training Random Forest Multicalss Classifier Model for UniProt P05106 using
Morgan Fingerprints features
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[512]:

```
PRINT(f'The results of Random Forest Multiclass Classifier model\nusing Morgan␣
  ↪Fingerprints features for UniProt P05106 are:')
print_dict_meaningful(rf_model_tuple_P05106_[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of Random Forest Multiclass Classifier model
using Morgan Fingerprints features for UniProt P05106 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.659
roc_auc_mean_score: 0.749
roc_auc_std_score: 0.015
precision: 0.666
recall: 0.659
f1_score: 0.661
confusion_matrix: [[5, 2, 1, 0, 0], [5, 280, 123, 3, 0], [3, 159, 301, 3, 1],
[0, 1, 1, 2, 0], [0, 0, 2, 1, 2]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

XGBoost Multiclass Classifier Model for P05106 with added Morgan Fingerprints Features

[514]:

```
weight_dict = 'balanced'

xgb_model_tuple_P05106_ = GenerateXGBoostModel(df=P05106_df_for_training__,␣
  ↪weight_dict=weight_dict, if_binary=False, bootstrap=True)
PRINT(f'Done training XGBoost Multicalss Classifier Model for UniProt P05106␣
  ↪using Morgan Fingerprints features')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.25      0.62      0.36         8
           1       0.66      0.74      0.69       411
           2       0.75      0.64      0.69       467
           3       0.17      0.50      0.25         4
           4       0.33      0.40      0.36         5

    accuracy                           0.68       895
   macro avg       0.43      0.58      0.47       895
weighted avg       0.70      0.68      0.68       895
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 0.756, Std Dev ROC AUC: 0.014
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done training XGBoost Multicalss Classifier Model for UniProt P05106 using
Morgan Fingerprints features
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[515]:

```
PRINT(f'The results of XGBoost Multiclass Classifier model\nusing Morgan␣
  ↪Fingerprints features for UniProt P05106 are:')
print_dict_meaningful(xgb_model_tuple_P05106_[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of XGBoost Multiclass Classifier model
using Morgan Fingerprints features for UniProt P05106 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.680
roc_auc_mean_score: 0.756
roc_auc_std_score: 0.014
precision: 0.698
recall: 0.680
f1_score: 0.685
confusion_matrix: [[5, 1, 2, 0, 0], [9, 303, 93, 4, 2], [6, 157, 297, 5, 2], [0,
1, 1, 2, 0], [0, 0, 2, 1, 2]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Pickup the best model trained so far for UniProt P05106

Visualize Trained Models Preformances

[555]:

```
df_for_preformance_comparison = pd.DataFrame({
    'RM_RDKit': list(rf_model_tuple_P05106[1].values()),
    'XGB_RDKit': list(xgb_model_tuple_P05106[1].values()),
    'RM_MF': list(rf_model_tuple_P05106_[1].values()),
    'XGB_MF': list(xgb_model_tuple_P05106_[1].values())
}, index=rf_model_tuple_P05106[1].keys())

df_for_preformance_comparison.head(7)
```

[555]:

```
                                                             RM_RDKit  \
accuracy                                                     0.631285
roc_auc_mean_score                                           0.741619
roc_auc_std_score                                            0.015577
```

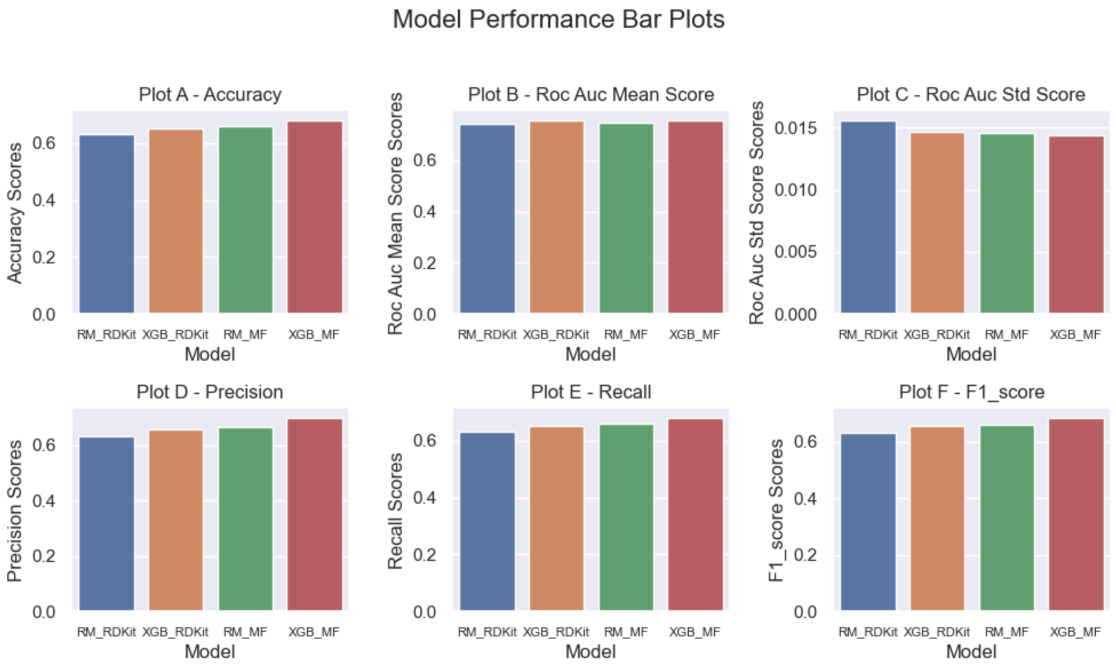
```
precision                                                     0.632359
recall                                                        0.631285
f1_score                                                      0.631454
confusion_matrix    [[4, 3, 1, 0, 0], [5, 247, 154, 0, 5], [0, 152…

                                                             XGB_RDKit  \
accuracy                                                      0.652514
roc_auc_mean_score                                             0.75593
roc_auc_std_score                                             0.014682
precision                                                     0.657788
recall                                                        0.652514
f1_score                                                      0.654542
confusion_matrix    [[5, 2, 1, 0, 0], [7, 264, 136, 1, 3], [1, 148…

                                                                 RM_MF  \
accuracy                                                      0.659218
roc_auc_mean_score                                            0.748679
roc_auc_std_score                                             0.014589
precision                                                     0.666021
recall                                                        0.659218
f1_score                                                      0.660872
confusion_matrix    [[5, 2, 1, 0, 0], [5, 280, 123, 3, 0], [3, 159…

                                                                XGB_MF
accuracy                                                      0.680447
roc_auc_mean_score                                            0.756037
roc_auc_std_score                                             0.014379
precision                                                     0.698349
recall                                                        0.680447
f1_score                                                      0.684672
confusion_matrix    [[5, 1, 2, 0, 0], [9, 303, 93, 4, 2], [6, 157,…
```

[556]:
```
df_for_preformance_comparison.drop(df_for_preformance_comparison.index[-1],␣
 ↪inplace=True)
df_for_preformance_comparison.reset_index(drop=True, inplace=True)
```

[557]:
```
df_for_preformance_comparison.head(6)
```

[557]:
```
   RM_RDKit XGB_RDKit     RM_MF    XGB_MF
0  0.631285  0.652514  0.659218  0.680447
1  0.741619   0.75593  0.748679  0.756037
2  0.015577  0.014682  0.014589  0.014379
3  0.632359  0.657788  0.666021  0.698349
4  0.631285  0.652514  0.659218  0.680447
5  0.631454  0.654542  0.660872  0.684672
```

[558]:
```
plot_model_comparison_countplots(df=df_for_preformance_comparison)
```



Choose & Save the Best Model By examining the performance plots of the models, it is evident that all models achieved high performances in accuracy, roc auc mean score, precision, recall, and f1 score. Additionally, all models obtained a roc auc std score below `0.015`, which is considered good. However, it is notable that the red bar, representing the *XGBoost* multiclass classifier model using Morgan fingerprints features, obtained the highest scores and the lowest roc auc std score.

Therefore, we will select this model as the preferred choice for UniProt P05106.

```
[559]: path = os.path.join('trained models/Best Model of each UniProt', 'xgb_P05106_MF.
         ↪joblib')
       dump(xgb_model_tuple_P05106_[0], path)

       PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

GraphConvModel Multiclass Classifier Model for P05106

```
[650]: P05106_df_for_training.head(2)
```

```
[650]:                                               SMILES  \
       0  COC(=O)c1ccc(COC(=O)N[C@@H](CCC(=O)N2CCN(c3ccc…
       1  COCCOCCOCCOCC(=O)Nc1cc(C[C@H](NS(=O)(=O)c2cccc…
```

```
   NumericUniProtTargetLabels
0                           1
1                           1
```

[652]:

```
csv_dataset_P05106_for_GraphConv_path = os.path.join('data', 'csv Files for␣
↪DeepChem GraphConvModel', 'P05106_df_GCM.csv')
```

[832]:

```
P05106_df_for_training.to_csv(csv_dataset_P05106_for_GraphConv_path,␣
↪index=False)
```

Build and Train Graph Conv Model

[653]:

```
training_score_list = []
validation_score_list = []
cv_folds = 10

metrics = [dc.metrics.Metric(dc.metrics.roc_auc_score, np.mean,␣
↪mode='classification')]

featurizer = dc.feat.ConvMolFeaturizer()
tasks = ['NumericUniProtTargetLabels']
loader = dc.data.CSVLoader(tasks=tasks,
                           smiles_field='SMILES',
                           featurizer=featurizer)
dataset = loader.featurize(csv_dataset_P05106_for_GraphConv_path)
```

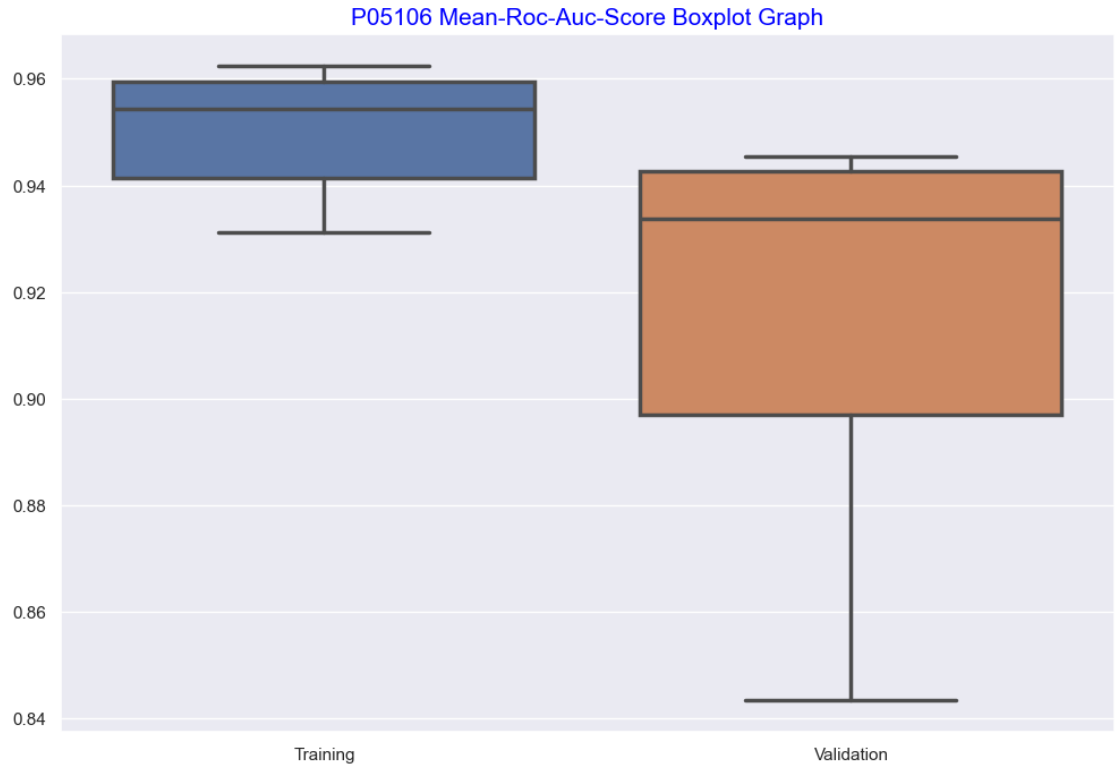
```
smiles_field is deprecated and will be removed in a future version of
DeepChem.Use feature_field instead.
```

[658]:

```
# Use splitter only once to obtain consistent train/valid splits
splitter = dc.splits.RandomSplitter()

# Create the model outside the loop
model = generate_graph_conv_model(dropout=0.2, batch_normalize=True,␣
↪n_classes=5, learning_rate=0.005, model_dir='models/gcm_P05106')

# Split the data into train&test, save 20% for testing & evaluation of the␣
↪trained model
train_dataset, test_dataset = splitter.train_test_split(dataset, test_size=0.2,␣
↪random_state=42)

# preforme cs_fold cross validation, in each iteration split the data into␣
↪train&val, train the model and test on the validation set.
for i in range(0, cv_folds):
    split = splitter.train_test_split(train_dataset)
    # Split the dataset into train, validation, and test sets
    train_dataset_, valid_dataset_ = split
    # Train the model
    model.fit(train_dataset_, nb_epoch=10)

    # Evaluate on training set
    train_scores = model.evaluate(train_dataset_, metrics, [], n_classes=5)
    training_score_list.append(train_scores['mean-roc_auc_score'])

    # Evaluate on validation set
    validation_scores = model.evaluate(valid_dataset_, metrics, [], n_classes=5)
    validation_score_list.append(validation_scores['mean-roc_auc_score'])
```

Visualize Model Preformance

```
[659]: GenerateBoxplotForModelPreformaceVisualization(UniProt='P05106',␣
        ↪cv_folds=cv_folds , training_score_list=training_score_list ,␣
        ↪validation_score_list=validation_score_list)
```



Predict on the Test Dataset and Visualize Performance

```
[661]: # Evaluate on test set
       test_scores = model.evaluate(test_dataset, metrics, [], n_classes=5)
       test_roc_auc = test_scores['mean-roc_auc_score']

       PRINT(f'Mean Roc Auc Score on the test dataset -> ({test_roc_auc:.3f})')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean Roc Auc Score on the test dataset -> (0.920)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Visualize Model Classification Report

```
[664]: true_labels = test_dataset.y.flatten()
       test_predictions = model.predict(test_dataset)


       # Get predicted label using helper function
       predicted_probs = get_class_labels(predicted_probs=test_predictions)

       report = classification_report(true_labels, predicted_probs)

       PRINT(report)
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00         9
         1.0       0.72      0.65      0.68       414
         2.0       0.72      0.78      0.75       461
         3.0       0.00      0.00      0.00         2
         4.0       0.69      1.00      0.82         9

    accuracy                           0.71       895
   macro avg       0.43      0.49      0.45       895
weighted avg       0.71      0.71      0.71       895

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

As observed, we achieved a high score in Roc Auc Mean, which is very promising and surpasses the performance of all the last four models we trained. However, the model's accuracy is not as impressive; the other four models exhibited higher accuracy when evaluated on the test set. Additionally, upon reviewing the classification report, it becomes apparent that the GraphConv model we trained struggled to identify the small classes in our imbalanced dataset, as evidenced by two rows of zeros in the classification report. Conversely, the XGBoost multiclass classifier model, selected as the best among the last four models, successfully classified some samples from the small classes (as evident in its classification report matrix).

In conclusion, we will adhere to our previous choice of the XGBoost model.

1.4.6 Models for P05107 Protein

```
[516]: P05107_df_for_training, P05107_df_with_uniprotes_col, mapped_label_dict_P05107␣
         ↪= generate_df_for_training_('P05107', 'P05107.csv', 'fourth_df_encoded.csv')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
P05107 model labels -> ['P11215', 'P20701']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Finished generating DataFrames for UniProt -> P05107.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[517]:

```
P05107_df_for_training.head(3)
```

[517]:

```
                                              SMILES  \
0  NCc1ccc(NC(=O)N2C(=O)CC2CC(=O)O)cc1.O=C(O)C(F)…
1  O=C(Nc1ccc(C(=O)Nc2cccc3cccc(S(=O)(=O)O)c23)cc…
2  COC(=O)CN1C(=O)S/C(=C\c2ccc(-c3ccc(C(=O)O)cc3)…

   NumericUniProtTargetLabels
0                           1
1                           0
2                           0
```

[518]:

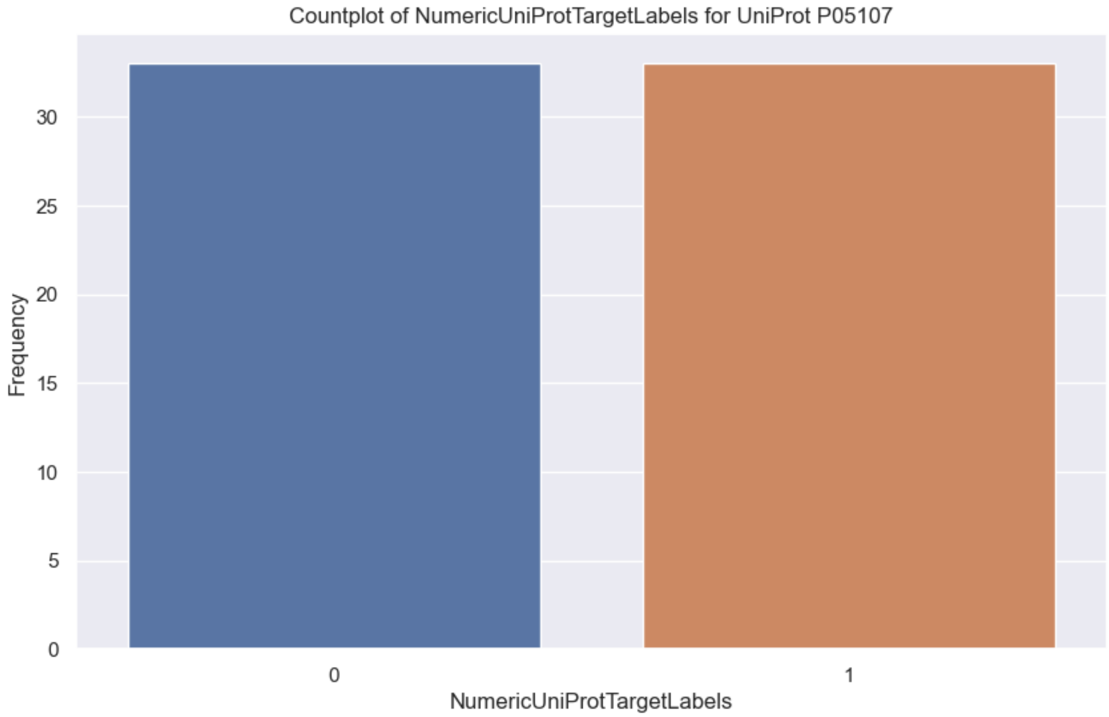
```
P05107_df_with_uniprotes_col.head(3)
```

[518]:

```
                                              SMILES UniProtTargetLabels  \
0  NCc1ccc(NC(=O)N2C(=O)CC2CC(=O)O)cc1.O=C(O)C(F)…              P20701
1  O=C(Nc1ccc(C(=O)Nc2cccc3cccc(S(=O)(=O)O)c23)cc…              P11215
2  COC(=O)CN1C(=O)S/C(=C\c2ccc(-c3ccc(C(=O)O)cc3)…              P11215

   NumericUniProtTargetLabels
0                           1
1                           0
2                           0
```

[519]:

```
plot_uniprot_numeric_label_frequency(df=P05107_df_for_training,␣
  ↪UniProt='P05107')
```



[520]:
```
PRINT(f'The mapped labels in ("UniProt": "index_label") format:
  ↪\n\n{mapped_label_dict_P05107}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The mapped labels in ("UniProt": "index_label") format:

{'P11215': 0, 'P20701': 1}
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Quick Dataset Analysis

- **Size of the data frame:** 66

- **Number of times each protein appears:**
  - P11215: 33
  - P20701: 33

Clearly, the dataset demonstrates perfect balance, obviating the necessity of assigning disparate weights to individual classes. Nonetheless, it is important to acknowledge the relatively diminutive size of the dataset. In light of this, the implementation of K-fold cross-validation in each model serves as a mitigating strategy for this limitation.

Furthermore, for a dataset of this modest size, opting for deep learning models such as the *GraphConvModel* from the DeepChem library may not be the most suitable choice. This is because the model could struggle to generalize effectively, leading to overfitting.

Consequently, we have decided to exclusively train Random Forest and XGBoost multiclass classifiers with balanced weights, selecting the superior performer from these two models..

Random Forest Multiclass Classifier Model for P05107 with added RDKitDescriptors Features

[521]:
```
P05107_df_for_training_ =␣
 ↪GenerateFeaturesByMoleculeSMILES(df=P05107_df_for_training)
```

[522]:
```
P05107_df_for_training_.head(3)
```

[522]:
```
                                              SMILES    MolWt  \
0  NCc1ccc(NC(=O)N2C(=O)CC2CC(=O)O)cc1.O=C(O)C(F)…  391.302
1  O=C(Nc1ccc(C(=O)Nc2cccc3cccc(S(=O)(=O)O)c23)cc…  491.481
2  COC(=O)CN1C(=O)S/C(=C\c2ccc(-c3ccc(C(=O)O)cc3)…  387.369

   NumValenceElectrons    TPSA  MolLogP  NumHeteroatoms  NumRotatableBonds  \
0                  148  150.03   1.3861              12                  4
1                  176  155.71   4.4993              11                  6
2                  138  114.12   2.8541               9                  5

   HeavyAtomCount  FractionCSP3  NumericUniProtTargetLabels
0              27      0.333333                           1
1              35      0.000000                           0
2              27      0.111111                           0
```

[523]:
```
weight_dict = {0: 1, 1: 1}

rf_model_tuple_P05107 = GenerateRandomForestModel(df=P05107_df_for_training_,␣
 ↪weight_dict=weight_dict, if_binary=True, bootstrap=True)
PRINT(f'Done training Random Forest Multicalss Classifier Model for UniProt␣
 ↪P05107 using RKDitDescriptors features')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.71      0.83         7
           1       0.78      1.00      0.88         7

    accuracy                           0.86        14
   macro avg       0.89      0.86      0.85        14
weighted avg       0.89      0.86      0.85        14

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 0.843, Std Dev ROC AUC: 0.093
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done training Random Forest Multicalss Classifier Model for UniProt P05107 using
RKDitDescriptors features
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[524]:

```
PRINT(f'The results of Random Forest Multiclass Classifier model\nusing␣
 ↪RKDitDescriptors features for nUniProt P05107 are:')
print_dict_meaningful(rf_model_tuple_P05107[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of Random Forest Multiclass Classifier model
using RKDitDescriptors features for nUniProt P05107 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.857
roc_auc_mean_score: 0.843
roc_auc_std_score: 0.093
precision: 0.889
recall: 0.857
f1_score: 0.854
confusion_matrix: [[5, 2], [0, 7]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

XGBoost Multiclass Classifier Model using RKDitDescriptors features for P05107

[525]:

```
weight_dict = {0: 1, 1: 1}

xgb_model_tuple_P05107 = GenerateXGBoostModel(df=P05107_df_for_training_,␣
 ↪weight_dict=weight_dict, if_binary=True, bootstrap=True)
PRINT(f'Done training XGBoost Multicalss Classifier Model for UniProt P05107␣
 ↪using RKDitDescriptors features')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.71      0.83         7
           1       0.78      1.00      0.88         7

    accuracy                           0.86        14
   macro avg       0.89      0.86      0.85        14
weighted avg       0.89      0.86      0.85        14

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 0.843, Std Dev ROC AUC: 0.093
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done training XGBoost Multicalss Classifier Model for UniProt P05107 using
```

```
RKDitDescriptors features
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[526]:

```
PRINT(f'The results of XGBoost Multiclass Classifier model\nusing␣
 ↪RDKitDescriptors features for UniProt P05107 are:')
print_dict_meaningful(xgb_model_tuple_P05107[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of XGBoost Multiclass Classifier model
using RDKitDescriptors features for UniProt P05107 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.857
roc_auc_mean_score: 0.843
roc_auc_std_score: 0.093
precision: 0.889
recall: 0.857
f1_score: 0.854
confusion_matrix: [[5, 2], [0, 7]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Random Forest Multiclass Classifier Model for P05107 with added Morgan Fingerprints Features

[527]:

```
P05107_df_for_training__ =␣
 ↪GenerateMorganFingerprintsFeaturesByMoleculeSMILES(df=P05107_df_for_training,␣
 ↪size=1024, radius=2)
```

[528]:

```
P05107_df_for_training__.head(2)
```

[528]:

```
                                              SMILES  \
0  NCc1ccc(NC(=O)N2C(=O)CC2CC(=O)O)cc1.O=C(O)C(F)…
1  O=C(Nc1ccc(C(=O)Nc2cccc3cccc(S(=O)(=O)O)c23)cc…

   NumericUniProtTargetLabels  Feature_0  Feature_1  Feature_2  Feature_3  \
0                           1        0.0        0.0        0.0        0.0
1                           0        0.0        0.0        0.0        0.0

   Feature_4  Feature_5  Feature_6  Feature_7  …  Feature_1014  \
0        0.0        0.0        0.0        0.0  …           0.0
1        0.0        0.0        0.0        0.0  …           0.0

   Feature_1015  Feature_1016  Feature_1017  Feature_1018  Feature_1019  \
0           0.0           0.0           0.0           0.0           1.0
1           0.0           0.0           0.0           0.0           0.0

   Feature_1020  Feature_1021  Feature_1022  Feature_1023
```

```
0           0.0           0.0           0.0           0.0
1           0.0           0.0           0.0           0.0

[2 rows x 1026 columns]
```

[529]:
```
weight_dict = {0: 1, 1: 1}

rf_model_tuple_P05107_ = GenerateRandomForestModel(df=P05107_df_for_training__,␣
  ↪weight_dict=weight_dict, if_binary=True, bootstrap=True)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.71      0.83         7
           1       0.78      1.00      0.88         7

    accuracy                           0.86        14
   macro avg       0.89      0.86      0.85        14
weighted avg       0.89      0.86      0.85        14

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 0.843, Std Dev ROC AUC: 0.093
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[530]:
```
PRINT(f'The results of Random Forest Multiclass Classifier model\nusing Morgan␣
  ↪Fingerprints features for UniProt P05107 are:')
print_dict_meaningful(rf_model_tuple_P05107_[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of Random Forest Multiclass Classifier model
using Morgan Fingerprints features for UniProt P05107 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.857
roc_auc_mean_score: 0.843
roc_auc_std_score: 0.093
precision: 0.889
recall: 0.857
f1_score: 0.854
confusion_matrix: [[5, 2], [0, 7]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

XGBoost Multiclass Classifier Model for P05107 with added Morgan Fingerprints Features

[531]:
```
weight_dict = {0: 1, 1: 1}
```

```
xgb_model_tuple_P05107_ = GenerateXGBoostModel(df=P05107_df_for_training__,␣
 ↪weight_dict=weight_dict, if_binary=True, bootstrap=True)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.71      0.83         7
           1       0.78      1.00      0.88         7

    accuracy                           0.86        14
   macro avg       0.89      0.86      0.85        14
weighted avg       0.89      0.86      0.85        14

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 0.843, Std Dev ROC AUC: 0.093
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[532]:

```
PRINT(f'The results of XGBoost Multiclass Classifier model\nusing Morgan␣
 ↪Fingerprints features for UniProt P05107 are:')
print_dict_meaningful(xgb_model_tuple_P05107_[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of XGBoost Multiclass Classifier model
using Morgan Fingerprints features for UniProt P05107 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.857
roc_auc_mean_score: 0.843
roc_auc_std_score: 0.093
precision: 0.889
recall: 0.857
f1_score: 0.854
confusion_matrix: [[5, 2], [0, 7]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Pickup the best model trained so far for UniProt P05107

Visualize Trained Models Preformances

[560]:

```
df_for_preformance_comparison = pd.DataFrame({
    'RM_RDKit': list(rf_model_tuple_P05107[1].values()),
    'XGB_RDKit': list(xgb_model_tuple_P05107[1].values()),
    'RM_MF': list(rf_model_tuple_P05107_[1].values()),
    'XGB_MF': list(xgb_model_tuple_P05107_[1].values())
}, index=rf_model_tuple_P05107[1].keys())

df_for_preformance_comparison.head(7)
```

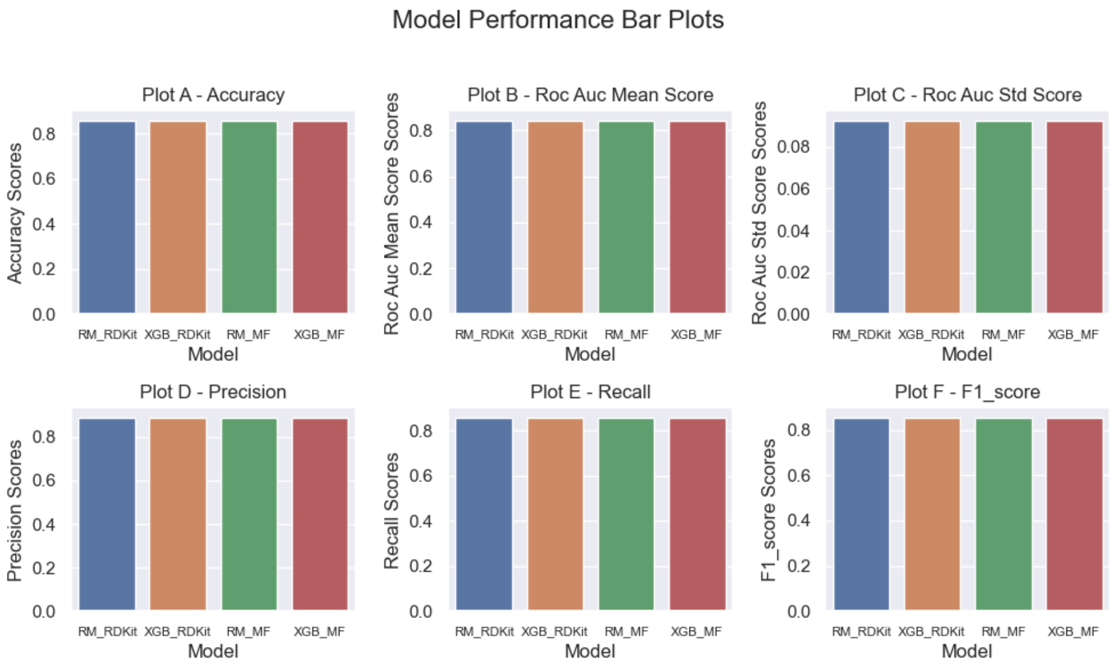
```
[560]:                           RM_RDKit          XGB_RDKit              RM_MF  \
       accuracy                   0.857143           0.857143           0.857143
       roc_auc_mean_score         0.842857           0.842857           0.842857
       roc_auc_std_score          0.092582           0.092582           0.092582
       precision                  0.888889           0.888889           0.888889
       recall                     0.857143           0.857143           0.857143
       f1_score                   0.854167           0.854167           0.854167
       confusion_matrix    [[5, 2], [0, 7]]  [[5, 2], [0, 7]]  [[5, 2], [0, 7]]

                                    XGB_MF
       accuracy                   0.857143
       roc_auc_mean_score         0.842857
       roc_auc_std_score          0.092582
       precision                  0.888889
       recall                     0.857143
       f1_score                   0.854167
       confusion_matrix    [[5, 2], [0, 7]]
```

```
[561]: df_for_preformance_comparison.drop(df_for_preformance_comparison.index[-1],␣
        ↪inplace=True)
       df_for_preformance_comparison.reset_index(drop=True, inplace=True)
```

```
[562]: df_for_preformance_comparison.head(6)
```

```
[562]:    RM_RDKit  XGB_RDKit     RM_MF    XGB_MF
       0  0.857143   0.857143  0.857143  0.857143
       1  0.842857   0.842857  0.842857  0.842857
       2  0.092582   0.092582  0.092582  0.092582
       3  0.888889   0.888889  0.888889  0.888889
       4  0.857143   0.857143  0.857143  0.857143
       5  0.854167   0.854167  0.854167  0.854167
```

```
[563]: plot_model_comparison_countplots(df=df_for_preformance_comparison)
```



Choose & Save the Best Model Upon revisiting the plots representing the performance of each model, it is evident that all models achieved identical results across all evaluation metrics. Such uniformity in results may be attributed to our small and unbalanced dataset.

As a result, we will randomly select a model. Our choice for the best model out of the four for UniProt P05107 will be th*e XGBoo*st multiclass classifier model, which utilized Morgan fingerprints as its features.

[564]:
```
path = os.path.join('trained models/Best Model of each UniProt', 'xgb_P05107_MF.
  ↪joblib')
dump(xgb_model_tuple_P05107_[0], path)

PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Pick the Best Model for P05107 Protein Based on the observations above, all four models performed quite well. Therefore, we will randomly select the XGBoost Multi-Class Classifier model that utilizes the generation of Morgan Fingerprints. The reason for this choice is that by utilizing Morgan Fingerprints features, we obtain 1024 new features for our data, while RDKit Descriptors utilize only 8 features (we select those 8 most meaningful features)

[463]:

```
xgb_P05107_morganf_path = 'trained models\\XGBoost Multiclass Classifier␣
 ↪Models\\xgb_model_P05107_.joblib'
xgb_P05107_morganf = load(xgb_P05107_morganf_path)

PRINT('Model Loaded.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Loaded.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[1019]:

```
xgb_model_filename = os.path.join('trained models/Best Model of each UniProt',␣
 ↪'final_xgb_P05107.joblib')
dump(xgb_P05107_morganf, xgb_model_filename)

PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.4.7 Models for P08648 Protein

[533]:

```
P08648_df_for_training, P08648_df_with_uniprotes_col, mapped_label_dict_P08648␣
 ↪= generate_df_for_training_('P08648', 'P08648.csv', 'fifth_df_encoded.csv')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
P08648 model labels -> ['P05556', 'P06756']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Finished generating DataFrames for UniProt -> P08648.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[534]:

```
P08648_df_for_training.head(3)
```

[534]:

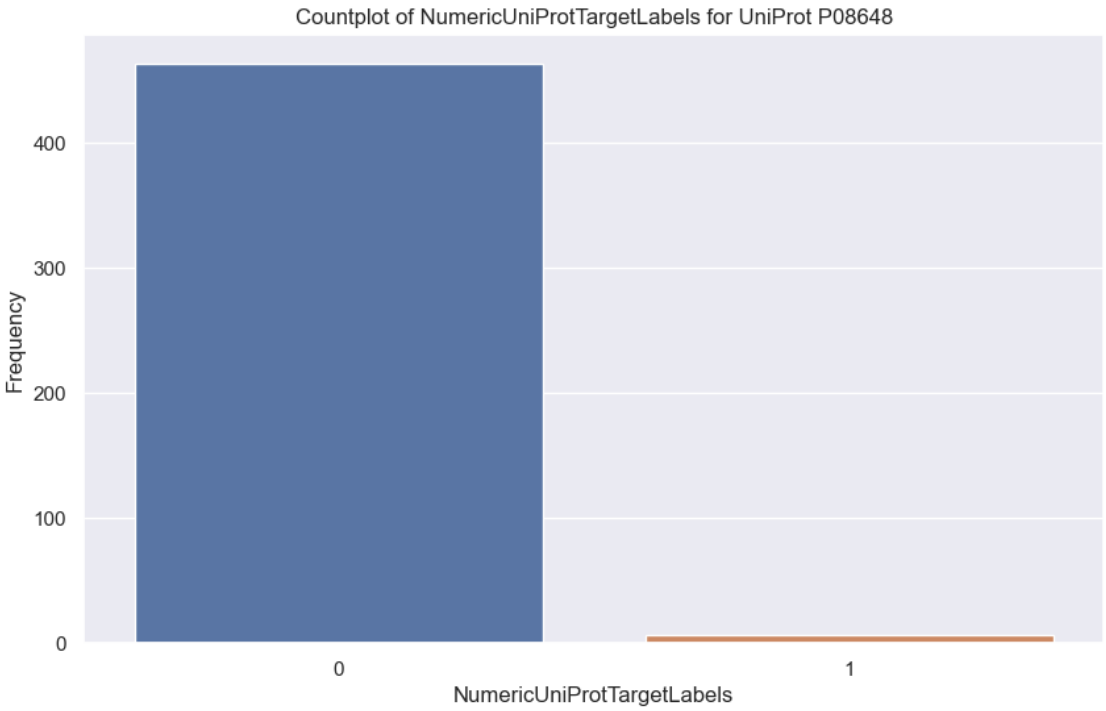
```
                                              SMILES  \
0  O=C(CO[C@@H]1C[C@@H](CNc2ccccn2)N(C(=O)OCc2ccc…
1  O=C(NCc1ccccc1)NC[C@H](NC(=O)[C@@H]1CCCN1S(=O)…
2  O=C(N[C@@H](Cc1cccc(OCCNc2ccccn2)c1)C(=O)O)c1c…

   NumericUniProtTargetLabels
0                           0
1                           0
2                           0
```

[535]:

```
P08648_df_with_uniprotes_col.head(3)
```

```
[535]:                                               SMILES UniProtTargetLabels  \
0  O=C(CO[C@@H]1C[C@@H](CNc2ccccn2)N(C(=O)OCc2ccc…              P05556
1  O=C(NCc1ccccc1)NC[C@H](NC(=O)[C@@H]1CCCN1S(=O)…              P05556
2  O=C(N[C@@H](Cc1cccc(OCCNc2ccccn2)c1)C(=O)O)c1c…              P05556

   NumericUniProtTargetLabels
0                           0
1                           0
2                           0
```

```
[536]: plot_uniprot_numeric_label_frequency(df=P08648_df_for_training, UniProt='P08648')
```


Countplot of NumericUniProtTargetLabels for UniProt P08648

```
[537]: PRINT(f'The mapped labels in ("UniProt": "index_label") format:\n\n{mapped_label_dict_P08648}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The mapped labels in ("UniProt": "index_label") format:

{'P05556': 0, 'P06756': 1}
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Check for Rows with NaN Values

[538]:

```
# Identify rows with 'float' values in the 'SMILES' column
float_rows = P08648_df_for_training['SMILES'].apply(lambda x: isinstance(x,␣
 ↪float))


# Display the rows with 'float' values
float_rows_data = P08648_df_for_training[float_rows]

PRINT(float_rows_data)
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    SMILES  NumericUniProtTargetLabels
387    NaN                           0
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[539]:

```
original_rows = P08648_df_for_training.shape[0]

P08648_df_for_training = P08648_df_for_training.dropna()

# Calculate the number of dropped rows
dropped_rows = original_rows - P08648_df_for_training.shape[0]

PRINT(f"{dropped_rows} rows were dropped.")
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
1 rows were dropped.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Quick Dataset Analysis

- **Size of the data frame:** 469 - 1 = 468

- **Number of times each protein appears:**
  - P05556: 463 - 1 = 462
  - P0756: 6

Once again, we are confronted with a relatively small dataset comprising only 468 rows. Consequently, opting for deep learning models like the *DeepChem GraphConvModel* might not be the optimal choice. In this scenario, we will adhere to employing *Random Forest* and *XGBoost* multiclass classifiers.

Furthermore, the dataset exhibits a notable imbalance, prompting us to explore potential solutions by assigning different weights to each class. This strategic approach aims to mitigate the impact of class imbalance during the training of our models.

Random Forest Multiclass Classifier Model for P08648 with added RDKitDescriptors Features

[540]:
```
P08648_df_for_training_ =␣
  ↪GenerateFeaturesByMoleculeSMILES(df=P08648_df_for_training)
```

[541]:
```
P08648_df_for_training_.head(3)
```

[541]:
```
                                              SMILES    MolWt  \
0  O=C(CO[C@@H]1C[C@@H](CNc2ccccn2)N(C(=O)OCc2ccc…  711.616
1  O=C(NCc1ccccc1)NC[C@H](NC(=O)[C@@H]1CCCN1S(=O)…  474.539
2  O=C(N[C@@H](Cc1cccc(OCCNc2ccccn2)c1)C(=O)O)c1c…  474.344

   NumValenceElectrons    TPSA  MolLogP  NumHeteroatoms  NumRotatableBonds  \
0                  268  159.19   4.3268              18                 13
1                  176  144.91   0.9085              11                  9
2                  166  100.55   4.3050               9                 10

   HeavyAtomCount  FractionCSP3  NumericUniProtTargetLabels
0              50      0.343750                           0
1              33      0.318182                           0
2              32      0.173913                           0
```

[542]:
```
weight_dict = 'balanced'

rf_model_tuple_P08648 = GenerateRandomForestModel(df=P08648_df_for_training_,␣
  ↪weight_dict=weight_dict, if_binary=True, bootstrap=True)
PRINT(f'Done training Random Forest Multicalss Classifier Model for UniProt␣
  ↪P08648 using RKDitDescriptors features')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99        93
           1       0.00      0.00      0.00         1

    accuracy                           0.98        94
   macro avg       0.49      0.49      0.49        94
weighted avg       0.98      0.98      0.98        94

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 0.495, Std Dev ROC AUC: 0.006
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done training Random Forest Multicalss Classifier Model for UniProt P08648 using
RKDitDescriptors features
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

It appears that even when attempting to assign balanced weights to both classes to address the issue of imbalanced data, challenges persist with the smaller class.

[543]:
```
PRINT(f'The results of the best Random Forest Multiclass Classifier␣
  ↪model\nusing RDKit Descriptors features for UniProt P05106 are:')
print_dict_meaningful(rf_model_tuple_P08648[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of the best Random Forest Multiclass Classifier model
using RDKit Descriptors features for UniProt P05106 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.979
roc_auc_mean_score: 0.495
roc_auc_std_score: 0.006
precision: 0.979
recall: 0.979
f1_score: 0.979
confusion_matrix: [[92, 1], [1, 0]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

XGBoost Multiclass Classifier Model using RKDitDescriptors features for P08648

[545]:
```
weight_dict = 'balanced'

xgb_model_tuple_P08648 = GenerateXGBoostModel(df=P08648_df_for_training_,␣
  ↪weight_dict=weight_dict, if_binary=True, bootstrap=True)
PRINT(f'Done training XGBoost Multicalss Classifier Model for UniProt P08648␣
  ↪using RKDitDescriptors features')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.95      0.97        93
           1       0.17      1.00      0.29         1

    accuracy                           0.95        94
   macro avg       0.58      0.97      0.63        94
weighted avg       0.99      0.95      0.97        94

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 0.974, Std Dev ROC AUC: 0.011
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done training XGBoost Multicalss Classifier Model for UniProt P08648 using
RKDitDescriptors features
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[546]: PRINT(f'The results of the best Random Forest Multiclass Classifier␣
         ↪model\nusing RDKit Descriptors features for UniProt P05106 are:')
       print_dict_meaningful(xgb_model_tuple_P08648[1])
       PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of the best Random Forest Multiclass Classifier model
using RDKit Descriptors features for UniProt P05106 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.947
roc_auc_mean_score: 0.974
roc_auc_std_score: 0.011
precision: 0.991
recall: 0.947
f1_score: 0.965
confusion_matrix: [[88, 5], [0, 1]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Random Forest Multiclass Classifier Model for P08648 with added Morgan Fingerprints Features

```
[547]: P08648_df_for_training__ =␣
         ↪GenerateMorganFingerprintsFeaturesByMoleculeSMILES(df=P08648_df_for_training,␣
         ↪size=1024, radius=2)
```

Drop new row that contains *Nan* values if existed

```
[548]: original_rows = P08648_df_for_training__.shape[0]

       P08648_df_for_training__ = P08648_df_for_training__.dropna()

       # Calculate the number of dropped rows
       dropped_rows = original_rows - P08648_df_for_training__.shape[0]

       PRINT(f"{dropped_rows} rows were dropped.")
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
2 rows were dropped.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[549]: P05106_df_for_training__.head(5)
```

```
[549]:                                               SMILES  \
        0  COC(=O)c1ccc(COC(=O)N[C@@H](CCC(=O)N2CCN(c3ccc…
        1  COCCOCCOCCOCC(=O)Nc1cc(C[C@H](NS(=O)(=O)c2cccc…
        2  Cl.O=C(NC(Cc1ccc2cc(OCCCN3CCNCC3)ccc2c1)C(=O)O…
        3                 CC(C)(C)c1nn2c(=O)cc(N3CCNCC3)nc2s1
```

```
4  O=C(O)C[C@@H](CC1CCN(C(=O)CCc2ccc3c(n2)NCCC3)C…

   NumericUniProtTargetLabels  Feature_0  Feature_1  Feature_2  Feature_3  \
0                           1        0.0        1.0        0.0        0.0
1                           1        0.0        1.0        0.0        0.0
2                           2        0.0        1.0        0.0        0.0
3                           2        0.0        0.0        0.0        0.0
4                           1        0.0        1.0        0.0        0.0

   Feature_4  Feature_5  Feature_6  Feature_7  …  Feature_1014  \
0        1.0        0.0        0.0        0.0  …           0.0
1        1.0        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        1.0        0.0        0.0        0.0  …           0.0

   Feature_1015  Feature_1016  Feature_1017  Feature_1018  Feature_1019  \
0           0.0           0.0           0.0           1.0           0.0
1           0.0           0.0           0.0           0.0           1.0
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           1.0           0.0           1.0

   Feature_1020  Feature_1021  Feature_1022  Feature_1023
0           0.0           0.0           0.0           0.0
1           0.0           0.0           0.0           0.0
2           0.0           0.0           0.0           0.0
3           0.0           0.0           0.0           0.0
4           0.0           0.0           0.0           0.0

[5 rows x 1026 columns]
```

[550]:

```
weight_dict = 'balanced'

rf_model_tuple_P08648_ = GenerateRandomForestModel(df=P08648_df_for_training__,␣
  ↪weight_dict=weight_dict, if_binary=True, bootstrap=True)
PRINT(f'Done training Random Forest Multicalss Classifier Model for UniProt␣
  ↪P08648 using Morgan Fingerprints features')
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      0.99      0.99        93
         1.0       0.50      1.00      0.67         1

    accuracy                           0.99        94
   macro avg       0.75      0.99      0.83        94
weighted avg       0.99      0.99      0.99        94
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 0.995, Std Dev ROC AUC: 0.005
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done training Random Forest Multicalss Classifier Model for UniProt P08648 using
Morgan Fingerprints features
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[551]:

```
PRINT(f'The results of Random Forest Multiclass Classifier model\nusing Morgan␣
 ↪Fingerprints features for UniProt P08648 are:')
print_dict_meaningful(rf_model_tuple_P08648_[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of Random Forest Multiclass Classifier model
using Morgan Fingerprints features for UniProt P08648 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.989
roc_auc_mean_score: 0.995
roc_auc_std_score: 0.005
precision: 0.995
recall: 0.989
f1_score: 0.991
confusion_matrix: [[92, 1], [0, 1]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

XGBoost Multiclass Classifier Model for P08648 with added Morgan Fingerprints Features

[552]:

```
weight_dict = 'balanced'

xgb_model_tuple_P08648_ = GenerateXGBoostModel(df=P08648_df_for_training__,␣
 ↪weight_dict=weight_dict, if_binary=True, bootstrap=True)
PRINT(f'Done training XGBoost Multicalss Classifier Model for UniProt P08648␣
 ↪using Morgan Fingerprints features')
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      0.99      0.99        93
         1.0       0.50      1.00      0.67         1

    accuracy                           0.99        94
   macro avg       0.75      0.99      0.83        94
weighted avg       0.99      0.99      0.99        94
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 0.995, Std Dev ROC AUC: 0.005
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done training XGBoost Multicalss Classifier Model for UniProt P08648 using
Morgan Fingerprints features
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[553]: PRINT(f'The results of the best XGBoost Multiclass Classifier model\nusing␣
         ↪Morgan Fingerprints features for UniProt P08648 are:')
       print_dict_meaningful(xgb_model_tuple_P08648_[1])
       PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of the best XGBoost Multiclass Classifier model
using Morgan Fingerprints features for UniProt P08648 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.989
roc_auc_mean_score: 0.995
roc_auc_std_score: 0.005
precision: 0.995
recall: 0.989
f1_score: 0.991
confusion_matrix: [[92, 1], [0, 1]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Pickup the best model trained so far for UniProt P08648

Visualize Trained Models Preformances

```
[565]: df_for_preformance_comparison = pd.DataFrame({
           'RM_RDKit': list(rf_model_tuple_P08648[1].values()),
           'XGB_RDKit': list(xgb_model_tuple_P08648[1].values()),
           'RM_MF': list(rf_model_tuple_P08648_[1].values()),
           'XGB_MF': list(xgb_model_tuple_P08648_[1].values())
       }, index=rf_model_tuple_P08648[1].keys())

       df_for_preformance_comparison.head(7)
```

```
[565]:                              RM_RDKit           XGB_RDKit               RM_MF  \
       accuracy                     0.978723            0.946809            0.989362
       roc_auc_mean_score           0.494516            0.973978            0.994731
       roc_auc_std_score            0.006129            0.010879            0.005267
       precision                    0.978723            0.991135            0.994681
       recall                       0.978723            0.946809            0.989362
       f1_score                     0.978723            0.965071            0.991106
       confusion_matrix     [[92, 1], [1, 0]]  [[88, 5], [0, 1]]  [[92, 1], [0, 1]]
```

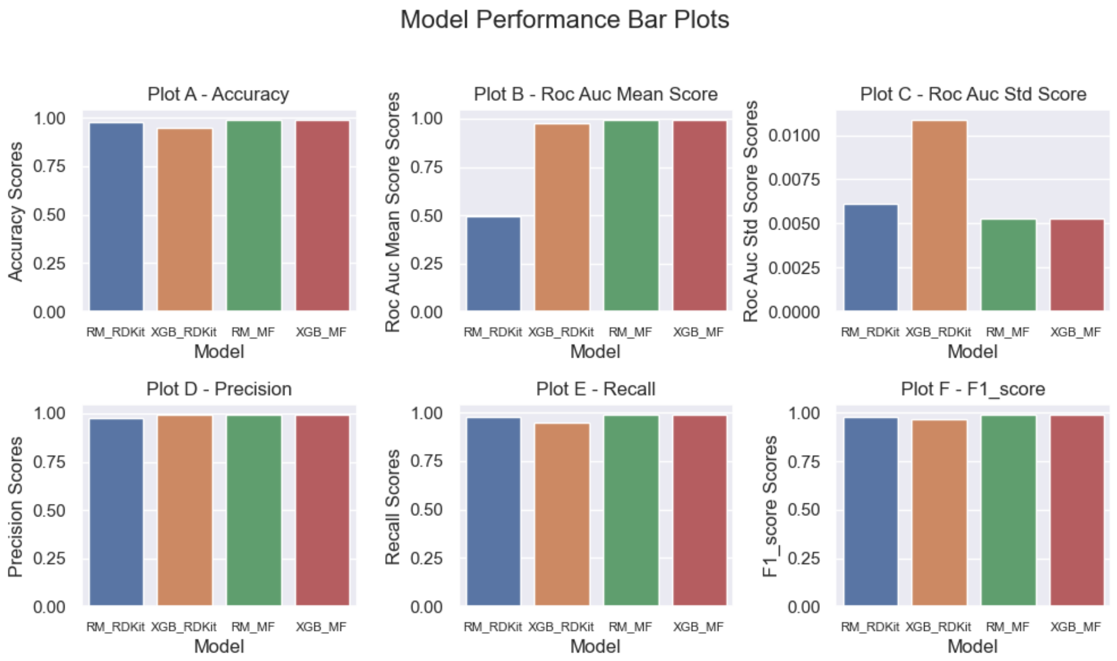
```
                              XGB_MF
accuracy                    0.989362
roc_auc_mean_score          0.994731
roc_auc_std_score           0.005267
precision                   0.994681
recall                      0.989362
f1_score                    0.991106
confusion_matrix    [[92, 1], [0, 1]]
```

```
[566]: df_for_preformance_comparison.drop(df_for_preformance_comparison.index[-1],␣
        ↪inplace=True)
       df_for_preformance_comparison.reset_index(drop=True, inplace=True)
```

```
[567]: df_for_preformance_comparison.head(6)
```

```
[567]:    RM_RDKit  XGB_RDKit     RM_MF    XGB_MF
       0  0.978723   0.946809  0.989362  0.989362
       1  0.494516   0.973978  0.994731  0.994731
       2  0.006129   0.010879  0.005267  0.005267
       3  0.978723   0.991135  0.994681  0.994681
       4  0.978723   0.946809  0.989362  0.989362
       5  0.978723   0.965071  0.991106  0.991106
```

```
[568]: plot_model_comparison_countplots(df=df_for_preformance_comparison)
```



Choose & Save the Best Model By analyzing the evaluation metrics performance plots for UniProt P08648, we observe that both the *Random Forest* and *XGBoost* models utilizing Morgan fingerprints features produced the best results.

Therefore, we will randomly select one model. Our choice will be the *Random Forest* as the best model for UniProt P08648 among the four models.

[587]:
```
path = os.path.join('trained models/Best Model of each UniProt', 'rf_P08648_MF.
 ↪joblib')
dump(rf_model_tuple_P08648_[0], path)

PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.4.8 Models for P17301 Protein

[570]:
```
P17301_df_for_training, P17301_df_with_uniprots_col, mapped_label_dict_P17301 =␣
 ↪generate_df_for_training_('P17301', 'P17301.csv', 'sixth_df_encoded.csv')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
P17301 model labels -> ['P05106', 'P05556']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Finished generating DataFrames for UniProt -> P17301.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[571]:
```
P17301_df_for_training.head(3)
```

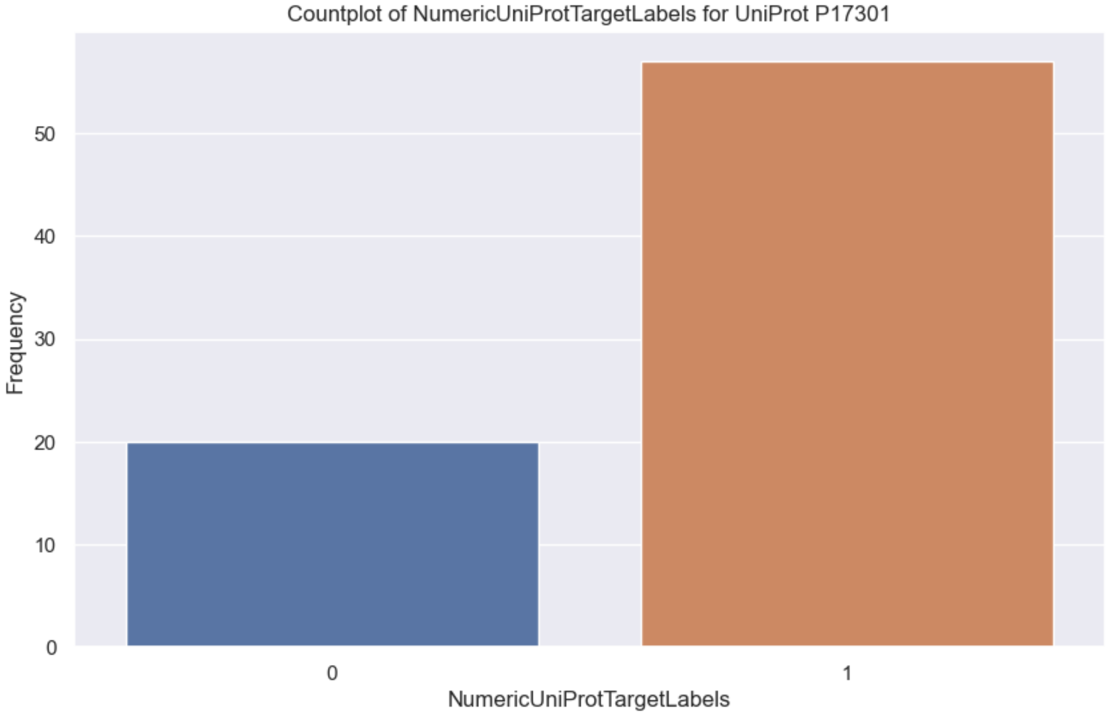
[571]:
```
                                              SMILES  \
0  O=C1N[C@H](C(=O)O)Cc2cccc(c2)OC/C=C/COc2ccc1c(…
1     O=C1N[C@H](C(=O)O)Cn2cc(nn2)CCCCOc2ccc(Cl)c1c2
2  CC(C)(C)c1cc(Br)cc([C@H](CC(=O)O)NC(=O)CNC(=O)…

   NumericUniProtTargetLabels
0                           1
1                           1
2                           1
```

[572]:
```
P17301_df_with_uniprots_col.head(3)
```

[572]:
```
                                              SMILES UniProtTargetLabels  \
0  O=C1N[C@H](C(=O)O)Cc2cccc(c2)OC/C=C/COc2ccc1c(…             P05556
1     O=C1N[C@H](C(=O)O)Cn2cc(nn2)CCCCOc2ccc(Cl)c1c2              P05556
2  CC(C)(C)c1cc(Br)cc([C@H](CC(=O)O)NC(=O)CNC(=O)…             P05556
```

```
   NumericUniProtTargetLabels
0                           1
1                           1
2                           1
```

[573]:
```
plot_uniprot_numeric_label_frequency(df=P17301_df_for_training,␣
↪UniProt='P17301')
```



[574]:
```
PRINT(f'The mapped labels in ("UniProt": "index_label") format:
↪\n\n{mapped_label_dict_P17301}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The mapped labels in ("UniProt": "index_label") format:

{'P05106': 0, 'P05556': 1}
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Quick Dataset Analysis

- **Size of the data frame:** 77

- **Number of occurrences for each protein:**
  - P05106: 20
  - P05556: 57

It appears that the sixth dataset is also characterized by its small size and a slight imbalance between the two classes. Consequently, we will once again refrain from training the *GraphConvModel* and concentrate solely on *Random Forest* and *XGBoost* multiclass classification models. We will employ balanced weights to address the dataset's imbalance and enhance model performance.

Random Forest Multiclass Classifier Model for P17301 with added RDKitDescriptors Features

```
[575]: P17301_df_for_training_ =␣
        ↪GenerateFeaturesByMoleculeSMILES(df=P17301_df_for_training)
```

```
[576]: P17301_df_for_training_.head(3)
```

```
[576]:                                                       SMILES    MolWt  \
       0  O=C1N[C@H](C(=O)O)Cc2cccc(c2)OC/C=C/COc2ccc1c(…  387.819
       1     O=C1N[C@H](C(=O)O)Cn2cc(nn2)CCCCOc2ccc(Cl)c1c2  364.789
       2  CC(C)(C)c1cc(Br)cc([C@H](CC(=O)O)NC(=O)CNC(=O)…  590.475

          NumValenceElectrons    TPSA  MolLogP  NumHeteroatoms  NumRotatableBonds  \
       0                  140   84.86   3.0931               7                  1
       1                  132  106.34   1.5298               9                  1
       2                  204  172.38   2.2462              12                  8

          HeavyAtomCount  FractionCSP3  NumericUniProtTargetLabels
       0              27      0.200000                           1
       1              25      0.375000                           1
       2              38      0.384615                           1
```

```
[577]: weight_dict = 'balanced'

       rf_model_tuple_P17301 = GenerateRandomForestModel(df=P17301_df_for_training_,␣
        ↪weight_dict=weight_dict, if_binary=True, bootstrap=True)
       PRINT(f'Done training Random Forest Multicalss Classifier Model for UniProt␣
        ↪P17301 using RKDitDescriptors features')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.75      0.86         4
           1       0.92      1.00      0.96        12

    accuracy                           0.94        16
   macro avg       0.96      0.88      0.91        16
weighted avg       0.94      0.94      0.93        16

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 0.885, Std Dev ROC AUC: 0.111
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done training Random Forest Multicalss Classifier Model for UniProt P17301 using
RKDitDescriptors features
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[103]:
```
PRINT(f'The results of Random Forest Multiclass Classifier model for\nUniProt␣
  ↪P17301 are:')
print_dict_meaningful(rf_model_tuple_P17301[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of Random Forest Multiclass Classifier model for
UniProt P17301 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.938
precision: 0.942
recall: 0.938
f1_score: 0.934
roc_auc_score: 0.875
confusion_matrix: [[3, 1], [0, 12]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Save Random Forest Multicalss Classifier Model for P17301

[536]:
```
rf_model_filename = os.path.join('trained models/Random Forest Multiclass␣
  ↪Classifier Models', 'rf_model_P17301.joblib')
dump(rf_model_tuple_P17301_01[0], rf_model_filename)

PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

XGBoost Multiclass Classifier Model using RKDitDescriptors features for P17301

[578]:
```
weight_dict = 'balanced'

xgb_model_tuple_P17301 = GenerateXGBoostModel(df=P17301_df_for_training_,␣
  ↪weight_dict=weight_dict, if_binary=True, bootstrap=True)
PRINT(f'Done training XGBoost Multicalss Classifier Model for UniProt P13612␣
  ↪using RKDitDescriptors features')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.75      0.86         4
           1       0.92      1.00      0.96        12
```

```
    accuracy                           0.94        16
   macro avg       0.96      0.88      0.91        16
weighted avg       0.94      0.94      0.93        16

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 0.885, Std Dev ROC AUC: 0.111
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done training XGBoost Multicalss Classifier Model for UniProt P13612 using
RKDitDescriptors features
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[588]:

```
PRINT(f'The results of XGBoost Multiclass Classifier model\nusing␣
 ↪RKDitDescriptors for UniProt P17301 are:')
print_dict_meaningful(xgb_model_tuple_P17301[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of XGBoost Multiclass Classifier model
using RKDitDescriptors for UniProt P17301 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.938
roc_auc_mean_score: 0.885
roc_auc_std_score: 0.111
precision: 0.942
recall: 0.938
f1_score: 0.934
confusion_matrix: [[3, 1], [0, 12]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Save XGBoost Multicalss Classifier Model for P17301

[539]:

```
xgb_model_filename = os.path.join('trained models/XGBoost Multiclass Classifier␣
 ↪Models', 'xgb_model_P17301.joblib')
dump(xgb_model_tuple_P17301_01[0], xgb_model_filename)

PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Random Forest Multiclass Classifier Model for P17301 with added Morgan Fingerprints Features

[580]:

```
P17301_df_for_training__ =␣
↪GenerateMorganFingerprintsFeaturesByMoleculeSMILES(df=P17301_df_for_training,␣
↪size=1024, radius=2)
```

```
[581]: P17301_df_for_training__
```

```
[581]:                                                SMILES  \
0   O=C1N[C@H](C(=O)O)Cc2cccc(c2)OC/C=C/COc2ccc1c(…
1      O=C1N[C@H](C(=O)O)Cn2cc(nn2)CCCCOc2ccc(Cl)c1c2
2   CC(C)(C)c1cc(Br)cc([C@H](CC(=O)O)NC(=O)CNC(=O)…
3   Cc1cccc(Cl)c1C(=O)N[C@@H](Cc1ccc(NC(=O)c2c(Cl)…
4   O=C(c1ccccc1)c1ccc([N-]S(=O)(=O)c2cccc(-c3ccc(…
..                                                …
72  O=C1N[C@H](C(=O)O)Cc2ccc(cc2)OC/C=C/COc2cccc(C…
73  O=C(O)CCNC(=O)c1cc(C(=O)Nc2ccc3c(c2)CNCC3)cc([…
74  COc1ccc(C(CC(=O)O)NC(=O)c2cc(C(=O)Nc3ccc4c(c3)…
75  Cc1cc(C)cc(S(=O)(=O)N2CCC[C@H]2C(=O)N[C@@H](CN…
76     O=C1N[C@H](C(=O)O)Cc2ccc(cc2)OCCCCOc2cccc(Cl)c21

    NumericUniProtTargetLabels  Feature_0  Feature_1  Feature_2  Feature_3  \
0                            1        0.0        0.0        0.0        0.0
1                            1        0.0        0.0        0.0        0.0
2                            1        0.0        1.0        0.0        0.0
3                            1        0.0        1.0        0.0        0.0
4                            1        0.0        0.0        0.0        0.0
..                         …        …        …        …        …
72                           1        0.0        0.0        0.0        0.0
73                           0        0.0        0.0        0.0        0.0
74                           0        0.0        1.0        0.0        0.0
75                           1        0.0        1.0        0.0        0.0
76                           1        0.0        0.0        0.0        0.0

    Feature_4  Feature_5  Feature_6  Feature_7  …  Feature_1014  \
0         0.0        0.0        0.0        0.0  …           0.0
1         1.0        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
..        …        …        …        …  …           …
72        0.0        0.0        0.0        0.0  …           0.0
73        0.0        0.0        0.0        0.0  …           0.0
74        0.0        0.0        0.0        0.0  …           0.0
75        1.0        0.0        0.0        0.0  …           0.0
76        1.0        0.0        0.0        0.0  …           0.0

    Feature_1015  Feature_1016  Feature_1017  Feature_1018  Feature_1019  \
0            0.0           0.0           0.0           0.0           1.0
```

```
1             0.0           0.0           0.0           0.0           1.0
2             0.0           0.0           0.0           0.0           1.0
3             0.0           0.0           0.0           0.0           0.0
4             0.0           0.0           0.0           0.0           0.0
..            …             …             …             …             …
72            0.0           0.0           0.0           0.0           1.0
73            0.0           0.0           0.0           0.0           0.0
74            0.0           0.0           0.0           0.0           0.0
75            0.0           0.0           0.0           0.0           1.0
76            0.0           0.0           0.0           0.0           1.0

    Feature_1020  Feature_1021  Feature_1022  Feature_1023
0            0.0           0.0           0.0           0.0
1            0.0           0.0           0.0           0.0
2            0.0           0.0           0.0           0.0
3            0.0           0.0           0.0           0.0
4            0.0           0.0           0.0           0.0
..            …             …             …             …
72           0.0           0.0           0.0           0.0
73           0.0           0.0           0.0           0.0
74           0.0           0.0           0.0           0.0
75           0.0           0.0           0.0           0.0
76           0.0           0.0           0.0           0.0

[77 rows x 1026 columns]
```

[583]:

```
weight_dict = 'balanced'

rf_model_tuple_P17301_ = GenerateRandomForestModel(df=P17301_df_for_training__,␣
 ↪weight_dict=weight_dict, if_binary=True, bootstrap=True)
PRINT(f'Done training Random Forest Multicalss Classifier Model for UniProt␣
 ↪P17301 using Morgan Fingerprints features')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       1.00      1.00      1.00        12

    accuracy                           1.00        16
   macro avg       1.00      1.00      1.00        16
weighted avg       1.00      1.00      1.00        16

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 1.000, Std Dev ROC AUC: 0.000
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done training Random Forest Multicalss Classifier Model for UniProt P17301 using
```

```
Morgan Fingerprints features
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[589]:

```
PRINT(f'The results of Random Forest Multiclass Classifier model\nusing Morgan␣
  ↪Fingerprints features for UniProt P17301 are:')
print_dict_meaningful(rf_model_tuple_P17301_[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of Random Forest Multiclass Classifier model
using Morgan Fingerprints features for UniProt P17301 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 1.000
roc_auc_mean_score: 1.000
roc_auc_std_score: 0.000
precision: 1.000
recall: 1.000
f1_score: 1.000
confusion_matrix: [[4, 0], [0, 12]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Save Random Forest Multicalss Classifier Model using Morgan Fingerprint features for P17301

[547]:

```
rf_model_filename = os.path.join('trained models/Random Forest Multiclass␣
  ↪Classifier Models', 'rf_model_P17301_.joblib')
dump(rf_model_tuple_P17301_01_[0], rf_model_filename)

PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

XGBoost Multiclass Classifier Model for P17301 with added Morgan Fingerprints Features

[585]:

```
weight_dict = 'balanced'

xgb_model_tuple_P17301_ = GenerateXGBoostModel(df=P17301_df_for_training__,␣
  ↪weight_dict=weight_dict, if_binary=True, bootstrap=True)
PRINT(f'Done training XGBoost Multicalss Classifier Model for UniProt P17301␣
  ↪using Morgan Fingerprints features')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       1.00      1.00      1.00        12
```

```
    accuracy                           1.00        16
   macro avg       1.00      1.00      1.00        16
weighted avg       1.00      1.00      1.00        16

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Mean ROC AUC: 1.000, Std Dev ROC AUC: 0.000
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done training XGBoost Multicalss Classifier Model for UniProt P17301 using
Morgan Fingerprints features
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[590]:

```
PRINT(f'The results of XGBoost Multiclass Classifier model\nusing Morgan␣
 ↪Fingerprints features for UniProt P17301 are:')
print_dict_meaningful(xgb_model_tuple_P17301_[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of XGBoost Multiclass Classifier model
using Morgan Fingerprints features for UniProt P17301 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 1.000
roc_auc_mean_score: 1.000
roc_auc_std_score: 0.000
precision: 1.000
recall: 1.000
f1_score: 1.000
confusion_matrix: [[4, 0], [0, 12]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Visualize Trained Models Preformances

[591]:

```
df_for_preformance_comparison = pd.DataFrame({
    'RM_RDKit': list(rf_model_tuple_P17301[1].values()),
    'XGB_RDKit': list(xgb_model_tuple_P17301[1].values()),
    'RM_MF': list(rf_model_tuple_P17301_[1].values()),
    'XGB_MF': list(xgb_model_tuple_P17301_[1].values())
}, index=rf_model_tuple_P17301[1].keys())

df_for_preformance_comparison.head(7)
```

[591]:

```
                    RM_RDKit   XGB_RDKit       RM_MF  \
accuracy              0.9375      0.9375         1.0
roc_auc_mean_score     0.885       0.885         1.0
roc_auc_std_score   0.111355    0.111355         0.0
precision           0.942308    0.942308         1.0
```

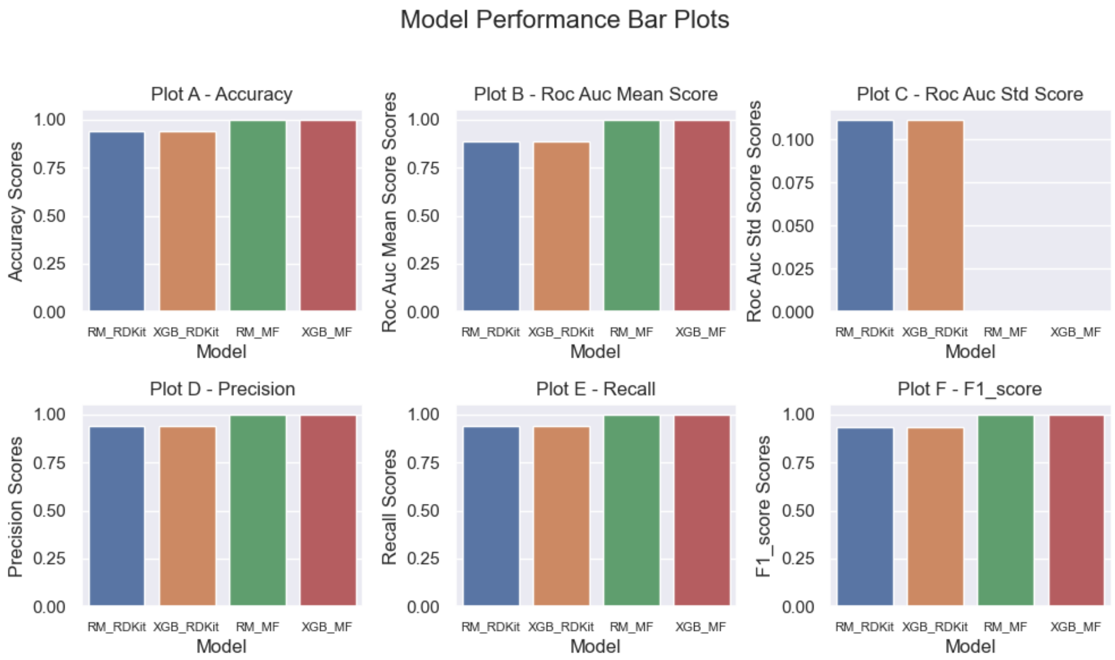
```
recall                          0.9375             0.9375                1.0
f1_score                      0.934286           0.934286                1.0
confusion_matrix    [[3, 1], [0, 12]]  [[3, 1], [0, 12]]  [[4, 0], [0, 12]]

                                XGB_MF
accuracy                           1.0
roc_auc_mean_score                 1.0
roc_auc_std_score                  0.0
precision                          1.0
recall                             1.0
f1_score                           1.0
confusion_matrix    [[4, 0], [0, 12]]
```

[592]:
```
# Drop index column
df_for_preformance_comparison.drop(df_for_preformance_comparison.index[-1],␣
 ↪inplace=True)

# Drop the last column (i.e., confusion_metrix column)
df_for_preformance_comparison.reset_index(drop=True, inplace=True)
```

[593]:
```
df_for_preformance_comparison.head(6)
```

[593]:
```
   RM_RDKit XGB_RDKit RM_MF XGB_MF
0    0.9375    0.9375   1.0    1.0
1     0.885     0.885   1.0    1.0
2  0.111355  0.111355   0.0    0.0
3  0.942308  0.942308   1.0    1.0
4    0.9375    0.9375   1.0    1.0
5  0.934286  0.934286   1.0    1.0
```

[594]:
```
plot_model_comparison_countplots(df=df_for_preformance_comparison)
```



```
[595]: path = os.path.join('trained models/Best Model of each UniProt', 'xgb_P17301_MF.
         ↪joblib')
       dump(xgb_model_tuple_P17301_[0], path)

       PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Pick the Best Model for P17301 Protein The result we obtained was as expected for our unbalanced dataset, despite our attempts to generalize it. We will choose the *XGBoost* model with *Morgan Fingerprints* features because it utilizes more hyperparameters during training to better generalize the model. This is necessary for our small and imbalanced dataset.

```
[556]: xgb_P17301_morganf_path = 'trained models\\XGBoost Multiclass Classifier␣
         ↪Models\\xgb_model_P17301_.joblib'
       xgb_P17301_morganf = load(xgb_P17301_morganf_path)
```

```
[557]: final_model_P17301 = os.path.join('trained models/Best Model of each UniProt',␣
         ↪'final_xbg_P17301.joblib')
       dump(xgb_P17301_morganf, final_model_P17301)

       PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.5 Make Prediction On Unseen Dataset for Final Results

Now that we have built, trained, and selected a model for each UniProt target dataset we need, we can generate real-time predictions using our best models.

To achieve this, we begin by extracting sub-datasets from our final dataframe on which we wish to execute predictions. For each sub-dataset, we will perform predictions using its corresponding model that we have built.

Finally, we will combine all the resulting data frames to obtain the final dataframe with the following columns: [SMILES, UniProtTarget, UniProtPartner].

```
[670]: final_df_path = 'data/dataset_for_prediction.csv'
```

```
[671]: f_df = pd.read_csv(final_df_path)

       PRINT(f'Loaded the final data frame')
       f_df.head(10)
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Loaded the final data frame
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[671]:                                               smiles uniprot_id1
       0  OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cccc2Cl)cc1)NC…      P13612
       1  C\C=C\[C@@H](CC(=O)O)NC(=O)C[C@@H](CC(C)C)NC(=…      P05556
       2  CN1[C@@H](CCCN=C(N)N)C(=O)NCC(=O)N[C@@H](CC(=O…      P05106
       3   OC(=O)C(CNC(=O)CCCCc1ccc2CCCNc2n1)c3cnc4ccccc4c3      P05106
       4  OC(=O)C[C@H](NC(=O)CN1CCC[C@@H](CCC2CCNCC2)C1=…      P05106
       5  N[C@@H](Cc1ccc(O)cc1)C(=O)N[C@H]2CSSC[C@H](NC(…      P05556
       6  CC(=O)N1CSC[C@@H]1C(=O)N[C@@H](Cc2ccc(OC(=O)C3…      P05556
       7  CC1CCC(C[C@H](NC(=O)[C@@H]2CCC(=O)N2Cc3ccccc3)…      P13612
       8  COP(=O)(O)[C@@H](CNC(=O)c1ccc(OCCC2CCNCC2)cc1)…      P05106
       9  NC(=N)Nc1cccc(c1)C(=O)Nc2ccc(CC(NS(=O)(=O)c3cc…      P05106
```

```
[672]: f_df.rename(columns={'uniprot_id1':'UniProtTarget', 'smiles':'SMILES'},␣
        ↪inplace=True)
```

```
[673]: f_df.head(3)
```

```
[673]:                                               SMILES UniProtTarget
       0  OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cccc2Cl)cc1)NC…        P13612
       1  C\C=C\[C@@H](CC(=O)O)NC(=O)C[C@@H](CC(C)C)NC(=…        P05556
       2  CN1[C@@H](CCCN=C(N)N)C(=O)NCC(=O)N[C@@H](CC(=O…        P05106
```

[675]:
```
# Verify that there is no duplicated rows in our prediction dataset, i.e.,␣
  ↪duplicated [SMILES, UniProtTarget] rows
duplicated_rows_df = f_df[f_df.duplicated()]

PRINT(f'Number of duplicated rows if the prediction dataframe ->␣
  ↪{duplicated_rows_df.shape[0]}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Number of duplicated rows if the prediction dataframe -> 0
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[676]:
```
target_dataframes = {}

# Iterate over unique UniProtTarget values
for target_value in f_df['UniProtTarget'].unique():
    # Filter the dataframe for the current UniProtTarget value
    target_df = f_df[f_df['UniProtTarget'] == target_value].copy()

    target_dataframes[target_value] = target_df
```

[677]:
```
target_dataframes.keys()
```

[677]:
```
dict_keys(['P13612', 'P05556', 'P05106', 'P05107', 'P08648', 'P17301'])
```

1.5.1 Predict for P13612

[678]:
```
P13612_label_dict = {0: 'P05556', 1: 'P26010'}
```

[679]:
```
P13612_pred = target_dataframes['P13612'].copy()

P13612_pred.head(5)
```

[679]:
```
                                               SMILES UniProtTarget
0   OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cccc2Cl)cc1)NC…        P13612
7   CC1CCC(C[C@H](NC(=O)[C@@H]2CCC(=O)N2Cc3ccccc3)…        P13612
10  CC(C)CCNC(=O)[C@@H]1OCO[C@H]1C(=O)N[C@@H](Cc2c…        P13612
14  OC(=O)CN(CC(=O)N[C@@H](Cc1ccc(OCc2c(Cl)cccc2Cl…        P13612
15  CCC\N=C/1\C(\C(=C1O)O)=N\[C@@H](Cc2ccc(OCc3c(C…        P13612
```

[680]:
```
P13612_pred = P13612_pred.reset_index(drop=True)

PRINT(f'Reseted the indexes of the data frame in order to avoid issues with␣
  ↪features generation')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Reseted the indexes of the data frame in order to avoid issues with features
```

```
generation
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[681]:
```
PRINT(f'Shape:\n\n{P13612_pred.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shape:

(1100, 2)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[682]:
```
P13612_pred.drop(['UniProtTarget'],axis=1, inplace=True)
```

[683]:
```
P13612_pred['TempColumnForModelTask'] = 0

PRINT(f'Added dummy column for modele "tasks" variable in order to compile the␣
 ↪model, later going to remove the column')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Added dummy column for modele "tasks" variable in order to compile the model,
later going to remove the column
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[684]:
```
P13612_pred.head(5)
```

[684]:
```
                                              SMILES  TempColumnForModelTask
0  OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cccc2Cl)cc1)NC…                        0
1  CC1CCC(C[C@H](NC(=O)[C@@H]2CCC(=O)N2Cc3ccccc3)…                        0
2  CC(C)CCNC(=O)[C@@H]1OCO[C@H]1C(=O)N[C@@H](Cc2c…                        0
3  OC(=O)CN(CC(=O)N[C@@H](Cc1ccc(OCc2c(Cl)cccc2Cl…                        0
4  CCC\N=C/1\C(\C(=C1O)O)=N\[C@@H](Cc2ccc(OCc3c(C…                        0
```

[685]:
```
gcm_P13612 = dc.models.GraphConvModel(model_dir='models/gcm_P13612', n_tasks=1)
gcm_P13612.restore()

PRINT('Model Loaded')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Loaded
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[686]:
```
P13612_gc_pred_csv_path = 'data/csv Files for DeepChem GraphConvModel/
 ↪P13612_pred_gc.csv'
```

[1189]:
```
P13612_pred.to_csv(P13612_gc_pred_csv_path, index=False)
```

[687]:
```
featurizer = dc.feat.ConvMolFeaturizer()
tasks = ['TempColumnForModelTask']
loader = dc.data.CSVLoader(tasks=tasks,
```

```
                            smiles_field='SMILES',
                            featurizer=featurizer)

dataset = loader.featurize(P13612_gc_pred_csv_path)
```

```
smiles_field is deprecated and will be removed in a future version of
DeepChem.Use feature_field instead.
```

[689]:
```
# Generate predictions
predicted_probs = gcm_P13612.predict(dataset)
PRINT(f'Done Predicting !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done Predicting !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[691]:
```
PRINT(f'Visualize few predictions probabilities [label_0, label_1] :
  ↪\n\n{predicted_probs[:5]}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Visualize few predictions probabilities [label_0, label_1] :

[[[0.95164675 0.04835324]]

 [[0.9578747  0.0421253 ]]

 [[0.8789022  0.12109788]]

 [[0.90091395 0.09908604]]

 [[0.92662746 0.07337261]]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[692]:
```
predicted_labels = get_class_labels(predicted_probs)

PRINT(f'Converted to probs to labeles using helper function !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Converted to probs to labeles using helper function !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[696]:
```
PRINT(f'Visualze few predictions using helper function :\n\n{predicted_labels[:
  ↪30]}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Visualze few predictions using helper function :

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[697]:
```
predictions = predicted_labels
labeled_predictions = [P13612_label_dict[prediction] for prediction in␣
  ↪predictions]
```

[698]:
```
P13612_pred['PredictedUniProtPartner'] = labeled_predictions
P13612_pred.drop(['TempColumnForModelTask'], axis=1, inplace=True)

PRINT('Merged the Predictions !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Merged the Predictions !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[699]:
```
P13612_pred
```

[699]:
```
                                                   SMILES  \
0     OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cccc2Cl)cc1)NC…
1     CC1CCC(C[C@H](NC(=O)[C@@H]2CCC(=O)N2Cc3ccccc3)…
2     CC(C)CCNC(=O)[C@@H]1OCO[C@H]1C(=O)N[C@@H](Cc2c…
3     OC(=O)CN(CC(=O)N[C@@H](Cc1ccc(OCc2c(Cl)cccc2Cl…
4     CCC\N=C/1\C(\C(=C1O)O)=N\[C@@H](Cc2ccc(OCc3c(C…
…                                                   …
1095  COc1ccccc1c2ccc(C[C@H](NC(=O)C3(CCCC3)c4ccc[n+…
1096  COc1ccccc1c2ccc(C[C@H](NC(=O)C3(CCCC3)c4cncc5c…
1097  COc1ccccc1c2ccc(C[C@H](NC(=O)C3(CC=CC3)c4cccnc…
1098  OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cncc2Cl)cc1)NC…
1099  OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cncc2Cl)cc1)NC…

     PredictedUniProtPartner
0                     P05556
1                     P05556
2                     P05556
3                     P05556
4                     P05556
…                          …
1095                  P26010
1096                  P26010
1097                  P26010
1098                  P05556
1099                  P05556

[1100 rows x 2 columns]
```

Looking at the data frame, we observe that our model has succeeded in predicting the smaller class as well. This achievement is notable given that we trained the model on an unbalanced dataset!

[700]:
```
P13612_pred.to_csv(os.path.join('predictions','P13612_pred.csv'), index=False)
```

```
PRINT('Predictions Saved!')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Predictions Saved!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.5.2 Predict for P05556

[822]:
```
P05556_label_dict = {0: 'P05106', 1: 'P06756', 2:'P08648',
                      3: 'P13612', 4:'P17301', 5: 'P56199',
                     6: 'Q13797'}
```

[809]:
```
P05556_pred = target_dataframes['P05556'].copy()

P05556_pred.head(2)
```

[809]:
```
                                              SMILES UniProtTarget
1  C\C=C\[C@@H](CC(=O)O)NC(=O)C[C@@H](CC(C)C)NC(=…        P05556
5  N[C@@H](Cc1ccc(O)cc1)C(=O)N[C@H]2CSSC[C@H](NC(…        P05556
```

[810]:
```
PRINT(f'Shepe:\n\n{P05556_pred.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shepe:

(948, 2)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[811]:
```
P05556_pred = P05556_pred.reset_index(drop=True)

PRINT(f'Reseted the indexes of the data frame in order to avoid issues with␣
  ↪features generation')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Reseted the indexes of the data frame in order to avoid issues with features
generation
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[812]:
```
P05556_pred.drop(['UniProtTarget'],axis=1, inplace=True)
```

[813]:
```
# Genera copy of the data frame for our features
P05556_pred_ = P05556_pred.copy()

# Apply the `calculate_descriptors` method in order to generate 8 new features␣
  ↪for df
P05556_pred_['MolecularDescriptors'] = P05556_pred_['SMILES'].
  ↪apply(calculate_descriptors)
```

```
# Transfer the array at each row under the 'MolecularDescriptors' column into␣
 ↪column with their corresponding names & drop the colunn
P05556_pred_[['MolWt', 'NumValenceElectrons', 'TPSA', 'MolLogP',␣
 ↪'NumHeteroatoms', 'NumRotatableBonds', 'HeavyAtomCount', 'FractionCSP3']] =␣
 ↪pd.DataFrame(P05556_pred_['MolecularDescriptors'].tolist(),␣
 ↪index=P05556_pred_.index)
P05556_pred_.drop(columns=['MolecularDescriptors'], axis=1, inplace=True)

# Reorder the columns names so that the label column will be the last column in␣
 ↪df
P05556_pred_ = P05556_pred_[['SMILES', 'MolWt', 'NumValenceElectrons', 'TPSA',␣
 ↪'MolLogP', 'NumHeteroatoms', 'NumRotatableBonds', 'HeavyAtomCount',␣
 ↪'FractionCSP3']]

PRINT(f'Done generating features for prediction !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done generating features for prediction !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[814]:

```
P05556_pred_.head(2)
```

[814]:

```
                                              SMILES    MolWt  \
0  C\C=C\[C@@H](CC(=O)O)NC(=O)C[C@@H](CC(C)C)NC(=…  522.646
1  N[C@@H](Cc1ccc(O)cc1)C(=O)N[C@H]2CSSC[C@H](NC(…  538.692

   NumValenceElectrons    TPSA  MolLogP  NumHeteroatoms  NumRotatableBonds  \
0                  204  136.63  4.63812               9                 13
1                  198  170.85  1.16640              12                  7

   HeavyAtomCount  FractionCSP3
0              38      0.379310
1              36      0.583333
```

[815]:

```
PRINT(f'Shape after generating features using RKDirDescriptors feature␣
 ↪generation:\n\n{P05556_pred_.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shape after generating features using RKDirDescriptors feature generation:

(948, 9)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[816]:

```
rf_P05556 = load('trained models/Best Model of each UniProt/rf_P05556_RDKD.
 ↪joblib')
PRINT(f'Loaded Model Successfully !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
Loaded Model Successfully !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[817]: # Drop SMILES column for prediction
       df_for_model_P05556 = P05556_pred_.drop(['SMILES'], axis=1)
```

```
[818]: df_for_model_P05556.head(2)
```

```
[818]:      MolWt  NumValenceElectrons    TPSA  MolLogP  NumHeteroatoms  \
       0  522.646                  204  136.63  4.63812               9
       1  538.692                  198  170.85  1.16640              12

          NumRotatableBonds  HeavyAtomCount  FractionCSP3
       0                 13              38      0.379310
       1                  7              36      0.583333
```

```
[819]: # Generate predictions on unseen data
       predictions = rf_P05556.predict(df_for_model_P05556)

       PRINT(f'Finished predicting on unseen data.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Finished predicting on unseen data.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[824]: PRINT(f'Prediction shape: {predictions.shape}\n\nVisualize few predictions:
         ↪\n\n{predictions[:30]}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Prediction shape: (948,)

Visualize few predictions:

[3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

In analyzing our model predictions on unseen data, we observe that the model successfully predicts classes with significantly low and unbalanced distributions in the dataset on which we trained the model. This suggests that our model excels in generalizing to identify even the smaller classes and predicting their protein-protein interactions (PPI)

```
[825]: # Generate list holding UniProts instead their labeled classes
       labeled_predictions = [P05556_label_dict[prediction] for prediction in␣
         ↪predictions]
```

```
[826]: PRINT(f'Labeled prediction (UniProt):\n\n{labeled_predictions[:15]}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Labeled prediction (UniProt):
```

```
['P13612', 'P13612', 'P13612', 'P13612', 'P08648', 'P13612', 'P13612', 'P13612',
'P13612', 'P13612', 'P13612', 'P13612', 'P13612', 'P13612', 'P13612']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[827]: P05556_pred['PredictedUniProtPartner'] = labeled_predictions

       PRINT('Merged the Predictions !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Merged the Predictions !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[828]: P05556_pred.head(3)
```

```
[828]:                                              SMILES PredictedUniProtPartner
       0  C\C=C\[C@@H](CC(=O)O)NC(=O)C[C@@H](CC(C)C)NC(=…                  P13612
       1  N[C@@H](Cc1ccc(O)cc1)C(=O)N[C@H]2CSSC[C@H](NC(…                  P13612
       2  CC(=O)N1CSC[C@@H]1C(=O)N[C@@H](Cc2ccc(OC(=O)C3…                  P13612
```

```
[829]: P05556_pred.to_csv(os.path.join('predictions','P05556_pred.csv'), index=False)

       PRINT('Predictions Saved!')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Predictions Saved!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.5.3 Predict for P05106

```
[701]: P05106_label_dict = {0: 'P05556', 1: 'P06756', 2: 'P08514', 3: 'P17301', 4:␣
         ↪'26006'}
```

```
[703]: P05106_pred = target_dataframes['P05106'].copy()


       P05106_pred.head(2)
```

```
[703]:                                              SMILES UniProtTarget
       2  CN1[C@@H](CCCN=C(N)N)C(=O)NCC(=O)N[C@@H](CC(=O…        P05106
       3   OC(=O)C(CNC(=O)CCCCc1ccc2CCCNc2n1)c3cnc4ccccc4c3        P05106
```

```
[704]: P05106_pred = P05106_pred.reset_index(drop=True)

       PRINT(f'Reseted the indexes of the data frame in order to avoid issues with␣
         ↪features generation')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Reseted the indexes of the data frame in order to avoid issues with features
```

```
generation
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[705]:

```
PRINT(f'Shape:\n\n{P05106_pred.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shape:

(1727, 2)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[706]:

```
P05106_pred.drop(['UniProtTarget'],axis=1, inplace=True)
```

[707]:

```
P05106_pred_ =␣
 ↪GenerateMorganFingerprintsFeaturesByMoleculeSMILES(df=P05106_pred,␣
 ↪size=1024, radius=2)
PRINT(f'Done generating features for prediction !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done generating features for prediction !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[708]:

```
P05106_pred_.head(2)
```

[708]:

```
                                              SMILES  Feature_0  Feature_1  \
0  CN1[C@@H](CCCN=C(N)N)C(=O)NCC(=O)N[C@@H](CC(=O…        0.0        0.0
1   OC(=O)C(CNC(=O)CCCCc1ccc2CCCNc2n1)c3cnc4ccccc4c3        0.0        1.0

   Feature_2  Feature_3  Feature_4  Feature_5  Feature_6  Feature_7  \
0        0.0        0.0        0.0        0.0        0.0        1.0
1        0.0        0.0        1.0        0.0        0.0        0.0

   Feature_8  …  Feature_1014  Feature_1015  Feature_1016  Feature_1017  \
0        0.0  …           0.0           0.0           0.0           0.0
1        0.0  …           0.0           0.0           0.0           1.0

   Feature_1018  Feature_1019  Feature_1020  Feature_1021  Feature_1022  \
0           0.0           1.0           0.0           0.0           0.0
1           0.0           0.0           0.0           0.0           0.0

   Feature_1023
0           0.0
1           0.0

[2 rows x 1025 columns]
```

[709]:

```
PRINT(f'Shape after generating features using Morgan Fingerprints:
 ↪\n\n{P05106_pred_.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shape after generating features using Morgan Fingerprints:

(1727, 1025)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[780]:

```
xgb_P05106 = load('trained models/Best Model of each UniProt/xgb_P05106_MF.
  ↪joblib')
PRINT(f'Model Loaded Successfully !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Loaded Successfully !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[712]:

```
# Drop SMILES column for prediction
df_for_model_P05106 = P05106_pred_.drop(['SMILES'], axis=1)
```

[713]:

```
predictions = xgb_P05106.predict(df_for_model_P05106)

PRINT(f'Finished predicting on unseen data.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Finished predicting on unseen data.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[714]:

```
PRINT(f'Prediction shape: {predictions.shape}\n\nVisualize few predictions:
  ↪\n\n{predictions[:30]}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Prediction shape: (1727,)

Visualize few predictions:

[2 1 2 2 1 1 1 3 2 1 2 1 1 1 2 2 1 1 1 1 1 2 1 1 2 2 2 1 2 1]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[715]:

```
# Generate list holding UniProts instead their labeled classes
labeled_predictions = [P05106_label_dict[prediction] for prediction in␣
  ↪predictions]
```

[716]:

```
PRINT(f'Labeled predictions (UniProt):\n\n{labeled_predictions[:15]}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Labeled predictions (UniProt):

['P08514', 'P06756', 'P08514', 'P08514', 'P06756', 'P06756', 'P06756', 'P17301',
'P08514', 'P06756', 'P08514', 'P06756', 'P06756', 'P06756', 'P08514']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[717]: # Merge predictions with our data frame
       P05106_pred['PredictedUniProtPartner'] = labeled_predictions

       PRINT('Merged the Predictions !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Merged the Predictions !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[718]: P05106_pred.head(3)
```

```
[718]:                                                     SMILES PredictedUniProtPartner
        0  CN1[C@@H](CCCN=C(N)N)C(=O)NCC(=O)N[C@@H](CC(=O…                  P08514
        1   OC(=O)C(CNC(=O)CCCCc1ccc2CCCNc2n1)c3cnc4ccccc4c3                  P06756
        2  OC(=O)C[C@H](NC(=O)CN1CCC[C@@H](CCC2CCNCC2)C1=…                  P08514
```

```
[719]: P05106_pred.to_csv(os.path.join('predictions','P05106_pred.csv'), index=False)

       PRINT('Predictions Saved!')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Predictions Saved!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.5.4 Predict for P05107

```
[720]: P05107_label_dict = {0: 'P11215', 1: 'P20701'}
```

```
[721]: P05107_pred = target_dataframes['P05107'].copy()

       P05107_pred.head(2)
```

```
[721]:                                                      SMILES UniProtTarget
        28  OC(=O)[C@@H]1CCCN1c2cc(ccn2)c3ccc(Sc4ccc5OCCOc…        P05107
        47  CN([C@@H]1CCN(C1)c2cc(ccn2)c3ccc(Sc4ccc5OCCOc5…        P05107
```

```
[722]: PRINT(f'Shepe:\n\n{P05107_pred.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shepe:

(339, 2)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[723]: P05107_pred = P05107_pred.reset_index(drop=True)

       PRINT(f'Reseted the indexes of the data frame in order to avoid issues with␣
         ↪features generation')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Reseted the indexes of the data frame in order to avoid issues with features
generation
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[724]:
```
P05107_pred.drop(['UniProtTarget'],axis=1, inplace=True)
```

[725]:
```
P05107_pred_ =␣
  ↪GenerateMorganFingerprintsFeaturesByMoleculeSMILES(df=P05107_pred,␣
  ↪size=1024, radius=2)
PRINT(f'Done generating features for prediction !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done generating features for prediction !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[726]:
```
P05107_pred_.head(2)
```

[726]:
```
                                              SMILES  Feature_0  Feature_1  \
0  OC(=O)[C@@H]1CCCN1c2cc(ccn2)c3ccc(Sc4ccc5OCCOc…        0.0        1.0
1  CN([C@@H]1CCN(C1)c2cc(ccn2)c3ccc(Sc4ccc5OCCOc5…        0.0        1.0

   Feature_2  Feature_3  Feature_4  Feature_5  Feature_6  Feature_7  \
0        0.0        0.0        1.0        0.0        0.0        0.0
1        0.0        0.0        0.0        0.0        0.0        0.0

   Feature_8  …  Feature_1014  Feature_1015  Feature_1016  Feature_1017  \
0        0.0  …           0.0           0.0           0.0           0.0
1        0.0  …           0.0           0.0           0.0           1.0

   Feature_1018  Feature_1019  Feature_1020  Feature_1021  Feature_1022  \
0           0.0           1.0           0.0           0.0           0.0
1           0.0           1.0           0.0           0.0           0.0

   Feature_1023
0           0.0
1           0.0

[2 rows x 1025 columns]
```

[727]:
```
PRINT(f'Shape after generating features using Morgan Fingerprints:
  ↪\n\n{P05107_pred_.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shape after generating features using Morgan Fingerprints:

(339, 1025)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[728]: P05107_pred_.dropna(axis=0, inplace=True)
```

```
[729]: PRINT(f'Shape after dropping:\n\n{P05107_pred_.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shape after dropping:

(339, 1025)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[779]: xgb_P05107 = load('trained models/Best Model of each UniProt/xgb_P05107_MF.
         ↪joblib')
       PRINT(f'Model Loaded Successfully !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Loaded Successfully !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[732]: # Drop SMILES column for prediction
       df_for_model_P05107 = P05107_pred_.drop(['SMILES'], axis=1)
```

```
[733]: # Generate prediction on unseen data
       predictions = xgb_P05107.predict(df_for_model_P05107)

       PRINT(f'Finished predicting on unseen data.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Finished predicting on unseen data.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[734]: PRINT(f'Prediction shape: {predictions.shape}\n\nVisualize few predictions:
         ↪\n\n{predictions[:50]}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Prediction shape: (339,)

Visualize few predictions:

[1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 0 1 1 1 1 1 1 1 1 1 0]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

As we can see, our model indeed identifies both classes in the predictions! This is a positive outcome, demonstrating the model's ability to accurately discern between the specified classes.

```
[735]: # Generate list holding UniProts instead their labeled classes
       labeled_predictions = [P05107_label_dict[prediction] for prediction in␣
         ↪predictions]
```

```
[737]: PRINT(f'Labeled prediction (UniProt):\n\n{labeled_predictions[:10]}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Labeled prediction (UniProt):

['P20701', 'P20701', 'P20701', 'P20701', 'P11215', 'P20701', 'P20701', 'P20701',
'P20701', 'P20701']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[738]:

```
# Merge predictions with our data frame
P05107_pred['PredictedUniProtPartner'] = labeled_predictions

PRINT('Merged the Predictions !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Merged the Predictions !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[739]:

```
P05107_pred.head(2)
```

[739]:

```
                                              SMILES PredictedUniProtPartner
0  OC(=O)[C@@H]1CCCN1c2cc(ccn2)c3ccc(Sc4ccc5OCCOc…                  P20701
1  CN([C@@H]1CCN(C1)c2cc(ccn2)c3ccc(Sc4ccc5OCCOc5…                  P20701
```

[740]:

```
P05107_pred.to_csv(os.path.join('predictions','P05107_pred.csv'), index=False)

PRINT('Predictions Saved!')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Predictions Saved!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.5.5 Predict for P08648

[741]:

```
P08648_label_dict = {0: 'P05556', 1: 'P06756'}
```

[742]:

```
P08648_pred = target_dataframes['P08648'].copy()

P08648_pred.head(2)
```

[742]:

```
                                                SMILES UniProtTarget
245  Cc1cc(C)c(c(C)c1)S(=O)(=O)N[C@@H](CNC(=O)C2=NO…        P08648
289  OC(=O)[C@H](Cc1cccc(OCCNc2ccccn2)c1)NC(=O)c3c(…        P08648
```

[743]:

```
P08648_pred = P08648_pred.reset_index(drop=True)

PRINT(f'Reseted the indexes of the data frame in order to avoid issues with␣
↪features generation')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Reseted the indexes of the data frame in order to avoid issues with features
generation
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[744]:

```
PRINT(f'Visualize data frame shape: {P08648_pred.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Visualize data frame shape: (76, 2)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[745]:

```
P08648_pred.drop(['UniProtTarget'],axis=1, inplace=True)
```

[746]:

```
P08648_pred_ =␣
  ↪GenerateMorganFingerprintsFeaturesByMoleculeSMILES(df=P08648_pred,␣
  ↪size=1024, radius=2)
PRINT(f'Done generating features for prediction !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done generating features for prediction !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[747]:

```
P08648_pred_.head(2)
```

[747]:

```
                                              SMILES  Feature_0  Feature_1  \
0  Cc1cc(C)c(c(C)c1)S(=O)(=O)N[C@@H](CNC(=O)C2=NO…        0.0        1.0
1  OC(=O)[C@H](Cc1cccc(OCCNc2ccccn2)c1)NC(=O)c3c(…        0.0        1.0

   Feature_2  Feature_3  Feature_4  Feature_5  Feature_6  Feature_7  \
0        0.0        0.0        0.0        0.0        0.0        0.0
1        0.0        0.0        0.0        0.0        0.0        0.0

   Feature_8  …  Feature_1014  Feature_1015  Feature_1016  Feature_1017  \
0        0.0  …           0.0           0.0           0.0           0.0
1        0.0  …           0.0           0.0           0.0           0.0

   Feature_1018  Feature_1019  Feature_1020  Feature_1021  Feature_1022  \
0           1.0           1.0           0.0           1.0           0.0
1           0.0           0.0           0.0           0.0           0.0

   Feature_1023
0           0.0
1           0.0

[2 rows x 1025 columns]
```

[748]:

```
PRINT(f'Shape after generating features using Morgan Fingerprints:
  ↪\n\n{P08648_pred_.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shape after generating features using Morgan Fingerprints:

(76, 1025)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[751]:

```
rf_P08648 = load('trained models/Best Model of each UniProt/rf_P08648_MF.
  ↪joblib')
PRINT(f'Loaded Model Successfully !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Loaded Model Successfully !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[752]:

```
# Drop SMILES column for predictions
df_for_model_P08648 = P08648_pred_.drop(['SMILES'], axis=1)
```

[753]:

```
df_for_model_P08648.head(2)
```

[753]:

```
   Feature_0  Feature_1  Feature_2  Feature_3  Feature_4  Feature_5  \
0        0.0        1.0        0.0        0.0        0.0        0.0
1        0.0        1.0        0.0        0.0        0.0        0.0

   Feature_6  Feature_7  Feature_8  Feature_9  …  Feature_1014  \
0        0.0        0.0        0.0        0.0  …           0.0
1        0.0        0.0        0.0        0.0  …           0.0

   Feature_1015  Feature_1016  Feature_1017  Feature_1018  Feature_1019  \
0           0.0           0.0           0.0           1.0           1.0
1           0.0           0.0           0.0           0.0           0.0

   Feature_1020  Feature_1021  Feature_1022  Feature_1023
0           0.0           1.0           0.0           0.0
1           0.0           0.0           0.0           0.0

[2 rows x 1024 columns]
```

[754]:

```
# Generate predictions on unseen data
predictions = rf_P08648.predict(df_for_model_P08648)

PRINT(f'Finished predicting on unseen data.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Finished predicting on unseen data.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[755]:

```
PRINT(f'Prediction shape: {predictions.shape}\n\nVisualize few predictions:
  ↪\n\n{predictions[:30]}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Prediction shape: (76,)

Visualize few predictions:

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0.]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[756]:

```
predictions = [int(value) for value in predictions]

PRINT(predictions)
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

As expected, although we attempted to balance the model using techniques such as assigning weight to the smaller class, our dataset was severely unbalanced for generalization with only two classes:

- Number of times P05556 appears -> 463
- Number of times P0756 appears -> 6

As a result, the model performed as anticipated on our small and unbalanced dataset.

[757]:

```
# Generate list holding UniProts instead their labeled classes
labeled_predictions = [P08648_label_dict[prediction] for prediction in␣
 ↪predictions]
```

[758]:

```
# Merge predictions with our data frame
P08648_pred['PredictedUniProtPartner'] = labeled_predictions

PRINT('Merged the Predictions !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Merged the Predictions !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[759]:

```
P08648_pred.head(2)
```

[759]:

```
                                              SMILES PredictedUniProtPartner
0  Cc1cc(C)c(c(C)c1)S(=O)(=O)N[C@@H](CNC(=O)C2=NO…                  P05556
1  OC(=O)[C@H](Cc1cccc(OCCNc2ccccn2)c1)NC(=O)c3c(…                  P05556
```

[760]:

```
P08648_pred.to_csv(os.path.join('predictions','P08648_pred.csv'), index=False)

PRINT('Predictions Saved!')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Predictions Saved!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.5.6 Predict for P17301

[761]:
```
P17301_label_dict = {0: 'P05106', 1: 'P05556'}
```

[762]:
```
P17301_pred = target_dataframes['P17301'].copy()

P17301_pred
```

[762]:
```
                                                 SMILES UniProtTarget
1149    Cc1ccccc1S(=O)(=O)N[C@@H](CNC(=O)c2cocc2)C(=O)O        P17301
3327  CCc1cc(O)c2c(O)c3C(=O)c4c(O)cccc4C(=O)c3cc2c1C…        P17301
```

[763]:
```
P17301_pred = P17301_pred.reset_index(drop=True)

PRINT(f'Reseted the indexes of the data frame in order to avoid issues with␣
 ↪features generation')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Reseted the indexes of the data frame in order to avoid issues with features
generation
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[764]:
```
PRINT(f'Shape:\n\n{P17301_pred.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shape:

(2, 2)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[765]:
```
P17301_pred.drop(['UniProtTarget'],axis=1, inplace=True)
```

[766]:
```
P17301_pred_ =␣
 ↪GenerateMorganFingerprintsFeaturesByMoleculeSMILES(df=P17301_pred,␣
 ↪size=1024, radius=2)
PRINT(f'Done generating features for prediction !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done generating features for prediction !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[767]:
```
P17301_pred_
```

[767]:
```
                                              SMILES  Feature_0  Feature_1  \
0   Cc1ccccc1S(=O)(=O)N[C@@H](CNC(=O)c2cocc2)C(=O)O        0.0        1.0
1  CCc1cc(O)c2c(O)c3C(=O)c4c(O)cccc4C(=O)c3cc2c1C…        0.0        0.0

   Feature_2  Feature_3  Feature_4  Feature_5  Feature_6  Feature_7  \
0        0.0        0.0        0.0        0.0        0.0        0.0
1        0.0        0.0        0.0        0.0        0.0        0.0

   Feature_8  …  Feature_1014  Feature_1015  Feature_1016  Feature_1017  \
0        0.0  …           0.0           0.0           0.0           0.0
1        0.0  …           0.0           0.0           0.0           0.0

   Feature_1018  Feature_1019  Feature_1020  Feature_1021  Feature_1022  \
0           0.0           0.0           0.0           0.0           0.0
1           0.0           0.0           0.0           0.0           0.0

   Feature_1023
0           0.0
1           0.0

[2 rows x 1025 columns]
```

[768]:
```
PRINT(f'Shape after generating features using Morgan Fingerprints:
↪\n\n{P17301_pred_.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shape after generating features using Morgan Fingerprints:

(2, 1025)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[771]:
```
xgb_P17301 = load('trained models/Best Model of each UniProt/xgb_P17301_MF.
↪joblib')
PRINT(f'Model Loaded Successfully !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Loaded Successfully !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[772]:
```
# Drop the SMILES column for predictions
df_for_model_P17301 = P17301_pred_.drop(['SMILES'], axis=1)

df_for_model_P17301.head(2)
```

[772]:
```
   Feature_0  Feature_1  Feature_2  Feature_3  Feature_4  Feature_5  \
0        0.0        1.0        0.0        0.0        0.0        0.0
1        0.0        0.0        0.0        0.0        0.0        0.0
```

```
   Feature_6  Feature_7  Feature_8  Feature_9  …  Feature_1014  \
0        0.0        0.0        0.0        0.0  …           0.0
1        0.0        0.0        0.0        0.0  …           0.0

   Feature_1015  Feature_1016  Feature_1017  Feature_1018  Feature_1019  \
0           0.0           0.0           0.0           0.0           0.0
1           0.0           0.0           0.0           0.0           0.0

   Feature_1020  Feature_1021  Feature_1022  Feature_1023
0           0.0           0.0           0.0           0.0
1           0.0           0.0           0.0           0.0

[2 rows x 1024 columns]
```

[773]:

```
# Generate predictions on unseen data
predictions = xgb_P17301.predict(df_for_model_P17301)

PRINT(f'Finished predicting on unseen data.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Finished predicting on unseen data.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[774]:

```
PRINT(f'Prediction shape: {predictions.shape}\n\nVisualize predictions:
  ↪\n\n{predictions}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Prediction shape: (2,)

Visualize predictions:

[1 1]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[775]:

```
labeled_predictions = [P17301_label_dict[prediction] for prediction in␣
  ↪predictions]
```

[776]:

```
P17301_pred['PredictedUniProtPartner'] = labeled_predictions

PRINT('Merged the Predictions !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Merged the Predictions !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[777]:

```
P17301_pred
```

[777]:

```
                                          SMILES PredictedUniProtPartner
0   Cc1ccccc1S(=O)(=O)N[C@@H](CNC(=O)c2cocc2)C(=O)O                  P05556
```

```
1  CCc1cc(O)c2c(O)c3C(=O)c4c(O)cccc4C(=O)c3cc2c1C…                   P05556
```

```
[778]: P17301_pred.to_csv(os.path.join('predictions','P17301_pred.csv'), index=False)

       PRINT('Predictions Saved!')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Predictions Saved!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.6 Putting it all together

After generating a sub-data frame for each unique UniProt we built a model for, and predicting the interactions for every single dataset associated with a given molecule SMILE and the UniProt target of its partner, we can finally combine all the datasets into a single comprehensive data frame.

```
[830]: prediction_dir = 'predictions'
```

```
[831]: P13612_df = pd.read_csv(os.path.join(prediction_dir, 'P13612_pred.csv'))
```

```
[832]: P13612_df['UniProtTarget'] = 'P13612'
```

```
[833]: P13612_df.head(3)
```

```
[833]:
                                              SMILES PredictedUniProtPartner  \
0  OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cccc2Cl)cc1)NC…                   P05556
1  CC1CCC(C[C@H](NC(=O)[C@@H]2CCC(=O)N2Cc3ccccc3)…                   P05556
2  CC(C)CCNC(=O)[C@@H]1OCO[C@H]1C(=O)N[C@@H](Cc2c…                   P05556

  UniProtTarget
0        P13612
1        P13612
2        P13612
```

```
[834]: P05556_df = pd.read_csv(os.path.join(prediction_dir, 'P05556_pred.csv'))
       P05556_df['UniProtTarget'] = 'P05556'
       P05556_df.head(3)
```

```
[834]:
                                              SMILES PredictedUniProtPartner  \
0  C\C=C\[C@@H](CC(=O)O)NC(=O)C[C@@H](CC(C)C)NC(=…                   P13612
1  N[C@@H](Cc1ccc(O)cc1)C(=O)N[C@H]2CSSC[C@H](NC(…                   P13612
2  CC(=O)N1CSC[C@@H]1C(=O)N[C@@H](Cc2ccc(OC(=O)C3…                   P13612

  UniProtTarget
0        P05556
1        P05556
2        P05556
```

[838]:
```
P05107_df = pd.read_csv(os.path.join(prediction_dir, 'P05107_pred.csv'))
P05107_df['UniProtTarget'] = 'P05107'
P05107_df.head(3)
```

[838]:
```
                                              SMILES PredictedUniProtPartner  \
0  OC(=O)[C@@H]1CCCN1c2cc(ccn2)c3ccc(Sc4ccc5OCCOc…                  P20701
1  CN([C@@H]1CCN(C1)c2cc(ccn2)c3ccc(Sc4ccc5OCCOc5…                  P20701
2  CC(C)c1ccccc1Sc2ccc(cc2C(F)(F)F)c3cc(ncn3)N4CC…                  P20701

  UniProtTarget
0        P05107
1        P05107
2        P05107
```

[839]:
```
P05106_df = pd.read_csv(os.path.join(prediction_dir, 'P05106_pred.csv'))
P05106_df['UniProtTarget'] = 'P05106'
P05106_df.head(3)
```

[839]:
```
                                              SMILES PredictedUniProtPartner  \
0  CN1[C@@H](CCCN=C(N)N)C(=O)NCC(=O)N[C@@H](CC(=O…                  P08514
1    OC(=O)C(CNC(=O)CCCCc1ccc2CCCNc2n1)c3cnc4ccccc4c3                  P06756
2  OC(=O)C[C@H](NC(=O)CN1CCC[C@@H](CCC2CCNCC2)C1=…                  P08514

  UniProtTarget
0        P05106
1        P05106
2        P05106
```

[840]:
```
P08648_df = pd.read_csv(os.path.join(prediction_dir, 'P08648_pred.csv'))
P08648_df['UniProtTarget'] = 'P08648'
P08648_df.head(3)
```

[840]:
```
                                              SMILES PredictedUniProtPartner  \
0  Cc1cc(C)c(c(C)c1)S(=O)(=O)N[C@@H](CNC(=O)C2=NO…                  P05556
1  OC(=O)[C@H](Cc1cccc(OCCNc2ccccn2)c1)NC(=O)c3c(…                  P05556
2  CCCN(C(=O)CC(=O)O)C1=C(C)C[C@H](N([C@@H](C)c2c…                  P05556

  UniProtTarget
0        P08648
1        P08648
2        P08648
```

[841]:
```
P17301_df = pd.read_csv(os.path.join(prediction_dir, 'P17301_pred.csv'))
P17301_df['UniProtTarget'] = 'P17301'
P17301_df.head(3)
```

```
[841]:                                                SMILES PredictedUniProtPartner  \
       0    Cc1ccccc1S(=O)(=O)N[C@@H](CNC(=O)c2cocc2)C(=O)O                  P05556
       1  CCc1cc(O)c2c(O)c3C(=O)c4c(O)cccc4C(=O)c3cc2c1C…                  P05556

         UniProtTarget
       0        P17301
       1        P17301
```

1.6.1 Combine all Data Frames into One Whole Data Frame

```
[849]: df_list = [P13612_df, P05556_df, P05107_df, P05106_df, P08648_df, P17301_df]
```

```
[843]: combined_df = pd.concat(df_list, ignore_index=True)
```

```
[844]: combined_df
```

```
[844]:                                                   SMILES  \
       0     OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cccc2Cl)cc1)NC…
       1     CC1CCC(C[C@H](NC(=O)[C@@H]2CCC(=O)N2Cc3ccccc3)…
       2     CC(C)CCNC(=O)[C@@H]1OCO[C@H]1C(=O)N[C@@H](Cc2c…
       3     OC(=O)CN(CC(=O)N[C@@H](Cc1ccc(OCc2c(Cl)cccc2Cl…
       4     CCC\N=C/1\C(\C(=C1O)O)=N\[C@@H](Cc2ccc(OCc3c(C…
       …                                                …
       4187  Cc1cc(C)c(C(=O)N[C@@H](CNC(=O)CO[C@@H]2C[C@@H]…
       4188  Cc1cc(C)c(c(C)c1)S(=O)(=O)N[C@@H](CNC(=O)CO[C@…
       4189  OC(=O)[C@H](CNC(=O)CO[C@@H]1C[C@@H](CNc2ccccn2…
       4190    Cc1ccccc1S(=O)(=O)N[C@@H](CNC(=O)c2cocc2)C(=O)O
       4191  CCc1cc(O)c2c(O)c3C(=O)c4c(O)cccc4C(=O)c3cc2c1C…

            PredictedUniProtPartner UniProtTarget
       0                     P05556        P13612
       1                     P05556        P13612
       2                     P05556        P13612
       3                     P05556        P13612
       4                     P05556        P13612
       …                        …            …
       4187                  P05556        P08648
       4188                  P05556        P08648
       4189                  P05556        P08648
       4190                  P05556        P17301
       4191                  P05556        P17301

       [4192 rows x 3 columns]
```

```
[845]: new_order = ['SMILES', 'UniProtTarget', 'PredictedUniProtPartner']
```

```
[846]: combined_df = combined_df[new_order]
```

[848]:

```
combined_df.head(3)
```

[848]:

```
                                              SMILES UniProtTarget  \
0  OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cccc2Cl)cc1)NC…        P13612
1  CC1CCC(C[C@H](NC(=O)[C@@H]2CCC(=O)N2Cc3ccccc3)…        P13612
2  CC(C)CCNC(=O)[C@@H]1OCO[C@H]1C(=O)N[C@@H](Cc2c…        P13612

  PredictedUniProtPartner
0                  P05556
1                  P05556
2                  P05556
```

[850]:

```
combined_df.to_csv('prediction_df.csv', index=False)

PRINT('SAVED & DONE !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
SAVED & DONE !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.6.2 Verify Data Frame Shape

[851]:

```
old_df = pd.read_csv(os.path.join('data','dataset_for_prediction.csv'))
```

[852]:

```
PRINT(f'Shapes check:\n\n{old_df.shape}\n\nvs.\n\n{combined_df.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shapes check:

(4192, 2)

vs.

(4192, 3)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Everything seems fine with the shapes; the additional column is a result of appending the predicted UniProt partner column to our combined dataframe.

[853]:

```
PRINT(f'-------------------------------------------------------------')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-------------------------------------------------------------
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[ ]:

```

```