IC50_Determination_Submission

May 13, 2025

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import curve_fit
import warnings
import math
import seaborn as sns
from IPython.display import display
from scipy.optimize import curve_fit
import warnings
from typing import Tuple, Optional
from tqdm import tqdm
```

[ ]:
```
Harvard_df = pd.read_csv('data/HarvardDataSet.csv')
```

[ ]:
```
Harvard_subdf_IC50 = Harvard_df[["Cell Name", "Small Molecule Name", "Small Mol␣
↪Concentration (uM)", "Mean Normalized Growth Rate Inhibition Value"]]
```

[ ]:
```
class BatchEC50Calculator:
    def __init__(self):
        pass

    @staticmethod
    def four_param_logistic(x, bottom, top, logEC50, hill_slope):
        return bottom + (top - bottom) / (1 + 10**((logEC50 - np.log10(x)) *␣
↪hill_slope))


    def calculate_ec50_single(self, concentrations, responses):
        concentrations = np.array(concentrations)
        responses = np.array(responses)

        # Basic checks
        if (len(concentrations) < 4
            or np.any(np.isnan(concentrations))
            or np.any(np.isnan(responses))
            or np.any(concentrations <= 0)):  # log10 can't handle zero/negative
```

```
            return None, [None] * 4

        log_med_conc = np.log10(np.median(concentrations))

        p0 = [min(responses), max(responses), log_med_conc, 1.0]

        try:
            popt, _ = curve_fit(
                self.four_param_logistic,
                concentrations,
                responses,
                p0=p0,
                maxfev=10000)
            # Validate popt (must be all real numbers)
            if popt is None or any(np.isnan(popt)):
                return None, [None] * 4

            ec50 = 10 ** popt[2]  # Convert logEC50 to EC50
        except (RuntimeError, ValueError, TypeError, OverflowError):
            return None, [None] * 4

        return ec50, popt


    def calculate_all_ec50(self, df, min_points = 4):
        results = []

        grouped = df.groupby(["Cell Name", "Small Molecule Name"])
        print(f"Total unique cell-drug pairs: {len(grouped)}")

        for (cell, drug), group in tqdm(grouped, desc="Fitting curves"):
            concentrations = group["Small Mol Concentration (uM)"].values
            responses = group["Mean Normalized Growth Rate Inhibition Value"].
↪values

            # Drop NaNs or invalid values
            mask = ~np.isnan(concentrations) & ~np.isnan(responses)
            concentrations = concentrations[mask]
            responses = responses[mask]

            if len(concentrations) < min_points:
                continue

            ec50, _ = self.calculate_ec50_single(concentrations, responses)
            results.append({
                "Cell Name": cell,
                "Small Molecule Name": drug,
```

```
                "EC50 (uM)": ec50})

        return pd.DataFrame(results)
```

[ ]:
```
Harvard_subdf_IC50["Cell Name"] = Harvard_subdf_IC50["Cell Name"].str.strip()
Harvard_subdf_IC50["Small Molecule Name"] = Harvard_subdf_IC50["Small Molecule␣
 ↪Name"].str.strip()

print("Expected:", 35 * 34)
print("Actual:", Harvard_subdf_IC50.groupby(["Cell Name", "Small Molecule␣
 ↪Name"]).ngroups)
```

[ ]:
```
calculator = BatchEC50Calculator()
ec50_results = calculator.calculate_all_ec50(Harvard_subdf_IC50)
```

[ ]:
```
ec50_results
```

[ ]:
```
def plot_all_ic50_curves(df, calculator):
    # Group the data by Cell Name and Small Molecule Name
    grouped = df.groupby(["Cell Name", "Small Molecule Name"])

    plot_dict = {}

    # Loop through each group
    for (cell, drug), group in grouped:
        concentrations = group["Small Mol Concentration (uM)"].values
        responses = group["Mean Normalized Growth Rate Inhibition Value"].values

        # safety checks
        if np.any(np.isnan(concentrations)) or np.any(np.isnan(responses)):
            print(f"Skipping {cell} - {drug}: NaNs in data.")
            continue

        ec50, popt = calculator.calculate_ec50_single(concentrations, responses)

        # check if fitting failed
        if ec50 is None or popt is None or any(p is None for p in popt):
            print(f"Skipping {cell} - {drug}: invalid fit.")
            continue

        plot = plot_ic50_curve(
            concentrations=concentrations,
            responses=responses,
            popt=popt,
            title=f"{cell} - {drug}")

        plot_dict[(cell, drug)] = plot
```

```
    return plot_dict

def plot_ic50_curve(concentrations, responses, popt, title=""):
    concentrations = np.array(concentrations)
    responses = np.array(responses)

    # Generate smooth x-axis range from the minimum concentration
    x_range = np.logspace(np.log10(min(concentrations)), np.
 ↪log10(max(concentrations) * 10), 200)

    # Predict curve from fitted parameters
    y_fit = BatchEC50Calculator.four_param_logistic(x_range, *popt)

    # Compute IC50
    ec50 = 10 ** popt[2]

    # Create a new figure and axis
    fig, ax = plt.subplots(figsize=(7, 5))

    # Plot the observed data
    ax.semilogx(concentrations, responses, 'o', label='Observed Data')
    ax.semilogx(x_range, y_fit, '-', label='Fitted Curve', color='blue')

    # Plot IC50
    ax.axvline(ec50, color='red', linestyle='--', label=f'IC50   {ec50:.3f} µM')

    # Dynamically adjust the Y-axis range
    y_min = min(responses) - 0.1
    y_max = max(responses) + 0.1
    ax.set_ylim([y_min, y_max])

    # Labels and title
    ax.set_xlabel("Concentration (µM)")
    ax.set_ylabel("Normalized Growth Rate Inhibition")
    ax.set_title(title)
    ax.legend()

    ax.grid(True, which='both', linestyle='--', linewidth=0.5)
    fig.tight_layout()

    return fig
```

[ ]:
```
calculator = BatchEC50Calculator()
plot_dictionary = plot_all_ic50_curves(Harvard_subdf_IC50, calculator)
```

[ ]:
```
plot_for_bt20_abemaciclib = plot_dictionary[("T47D", "Torin2")]
display(plot_for_bt20_abemaciclib)
```

0.0.1 Some IC50 values don't make sense because they are out of range and being extrapolated... need to filter those out.

```
[ ]: condition1 = (ec50_results["EC50 (uM)"] < 10.0) & (ec50_results["EC50 (uM)"] >␣
      ↪0.001000)
     IC50s_within_range = ec50_results[condition1]
     print(f"Only {len(IC50s_within_range)} IC50 values are valid and within range␣
      ↪of testing concentrations. This is ~{math.ceil(100*len(IC50s_within_range) /␣
      ↪len(ec50_results))}% of the IC50 values.")
```

```
[ ]: condition2 = (ec50_results["EC50 (uM)"] < 10.0)
     IC50s_less_than10 = ec50_results[condition2]

     print(f"{len(IC50s_less_than10) - len(IC50s_within_range) } chemicals are␣
      ↪extremely potent.")
```

```
[ ]: extremely_potent = pd.concat([IC50s_less_than10, IC50s_within_range])
     extremely_potent = extremely_potent.drop_duplicates(keep=False)
```

```
[ ]: IC50s_within_range
```

```
[ ]: pivot_table = IC50s_within_range.pivot_table(
         index='Small Molecule Name',
         columns='Cell Name',
         values='EC50 (uM)')
     print(f"NOTE: Those with NaN values were filtered out. Their IC50 values were␣
      ↪out of range (i.e. extrapolated and estimated). Should be␣
      ↪{1190-IC50s_within_range.shape[0]} NaN values.")
     pivot_table
```

```
[ ]: pivot_table.isna().sum().sum() # Those with NaN values were filtered out. Their␣
      ↪IC50 values could not be reliably calculated.
```

```
[ ]: idx = IC50s_within_range.groupby("Cell Name")["EC50 (uM)"].idxmin()
     lowest_ic50_per_cell = IC50s_within_range.loc[idx].reset_index(drop=True)
     lowest_ic50_per_cell
```

```
[ ]: def plot_ic50drugs_per_cell_line(df):
         # Get unique cell lines and assign each to a number
         cell_lines = df["Cell Name"].unique()
         cell_to_x = {cell: i for i, cell in enumerate(cell_lines)}

         # Get unique drugs and assign a color to each
         drugs = df["Small Molecule Name"].unique()
         palette = sns.color_palette("hsv", len(drugs))  # Color palette for drugs
         drug_to_color = {drug: palette[i] for i, drug in enumerate(drugs)}
```

```
    plt.figure(figsize=(16, 7))

    # Plot each point with color coding by drug
    for _, row in df.iterrows():
        x = cell_to_x[row["Cell Name"]]
        y = row["EC50 (uM)"]
        drug = row["Small Molecule Name"]
        color = drug_to_color[drug]

        plt.scatter(x, y, color=color, s=30)

    # Set x-axis labels
    plt.xticks(ticks=range(len(cell_lines)), labels=cell_lines, rotation=90)

    # Use log scale for y-axis (IC50 values)
    plt.yscale("log")
    plt.ylabel("IC50 (uM) (log scale)")
    plt.xlabel("Cell Line")
    plt.title("Valid IC50 Values per Cell Line (Color-Coded by Drug)")

    # Legend for the drugs
    handles = [plt.Line2D([0], [0], marker='o', color='w', label=drug,␣
↪markerfacecolor=color, markersize=8) for drug, color in drug_to_color.
↪items()]
    plt.legend(handles=handles, title="Drug", bbox_to_anchor=(1.05, 1),␣
↪loc='upper left')
    plt.grid(True, which="both", linestyle="--", linewidth=0.5)
    plt.tight_layout()
    plt.show()
```

[ ]:
```
plot_ic50drugs_per_cell_line(IC50s_within_range)
```

[ ]:
```
# Subframes by cell line with good IC50 values
cell_line_subframes = {
    cell: group.reset_index(drop=True)
    for cell, group in IC50s_within_range.groupby("Cell Name")}
```

[ ]:
```
IC50s_within_range.to_csv("valid_IC50s_within_range.csv")
```

[ ]:
```
plot_for_bt20_abemaciclib = plot_dictionary[("T47D", "Torin2")]
display(plot_for_bt20_abemaciclib)
```

[ ]:
```
plot_for_bt20_abemaciclib = plot_dictionary[("CAL-51", "Taxol")]
display(plot_for_bt20_abemaciclib)
```

[ ]:
```
plot_dictionary[("CAL-120", "AZD7762")]
```

[ ]:
```
plot_dictionary[("HME1", "Ceritinib")]
```

[ ]:
```

```