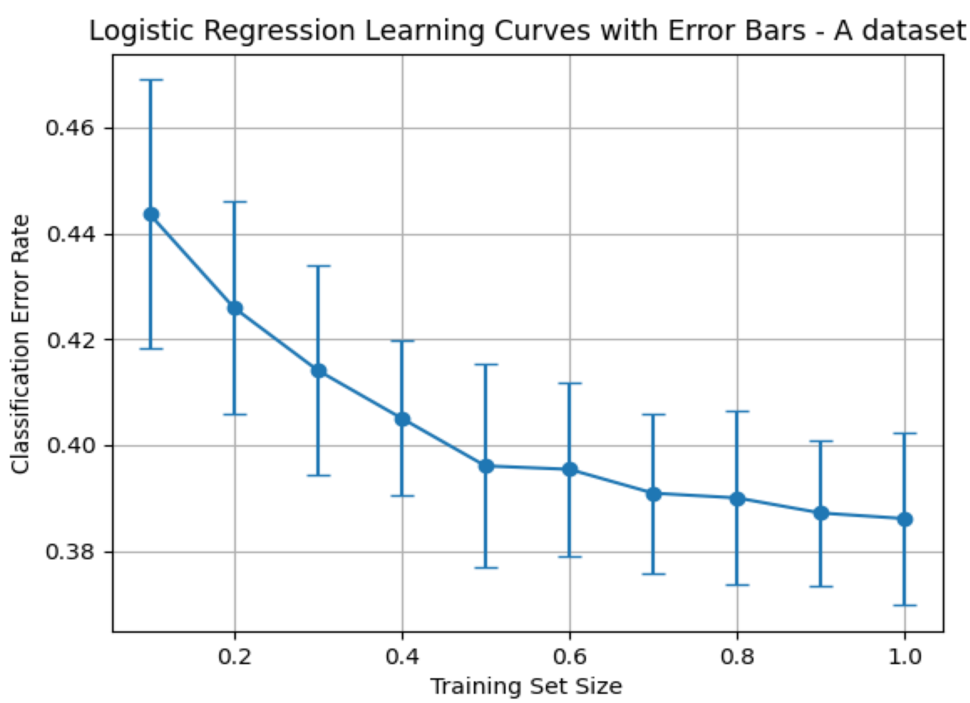
Sunilsakthivel Sakthi Velavan
Roni Khardon
CSCI-B 555
6 Nov.2023

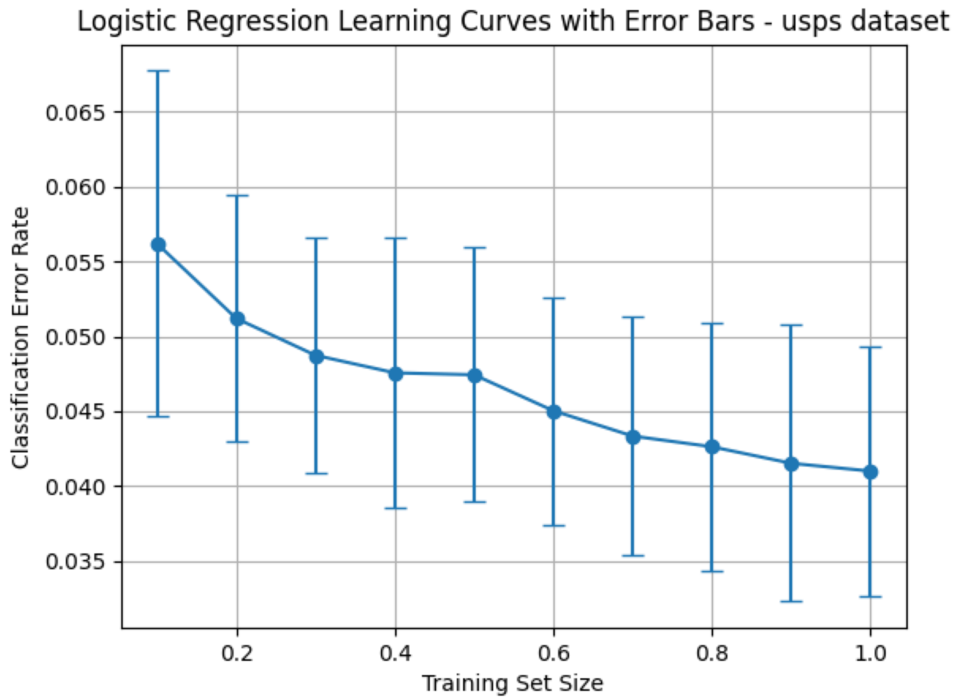
Programming Project 3

Task Overview:

This programming project you described involves implementing and evaluating a Bayesian Generalized Linear Model (GLM) with different likelihood models for classification and count prediction in Logistic, Poisson and Ordinal Regression. This is done with two functions in particular: The **glm** function performs GLM regression. It takes the input data X, target variable y, GLM type (Logistic, Poisson, or Ordinal), and other parameters such as regularization strength alpha, maximum iterations, and convergence tolerance. It also uses the Newton-Raphson method to estimate the GLM parameters. Then, the **glm_output_run** function iterates over GLM types and datasets to perform regression, assess performance, and visualize the learning curves. It first loads data and labels from CSV files based on the dataset and GLM type. It then iterates over a set number of runs (e.g., 30) and different training set sizes. For each run, it randomly splits the data into training and test sets and trains the GLM model on the training data and assesses its performance on the test data. Then, it calculates error rates and records the number of iterations used for convergence. It finally calculates and displays the mean error rates and standard deviations for each training set size and plots learning curves with error bars.

Logistic Regression:

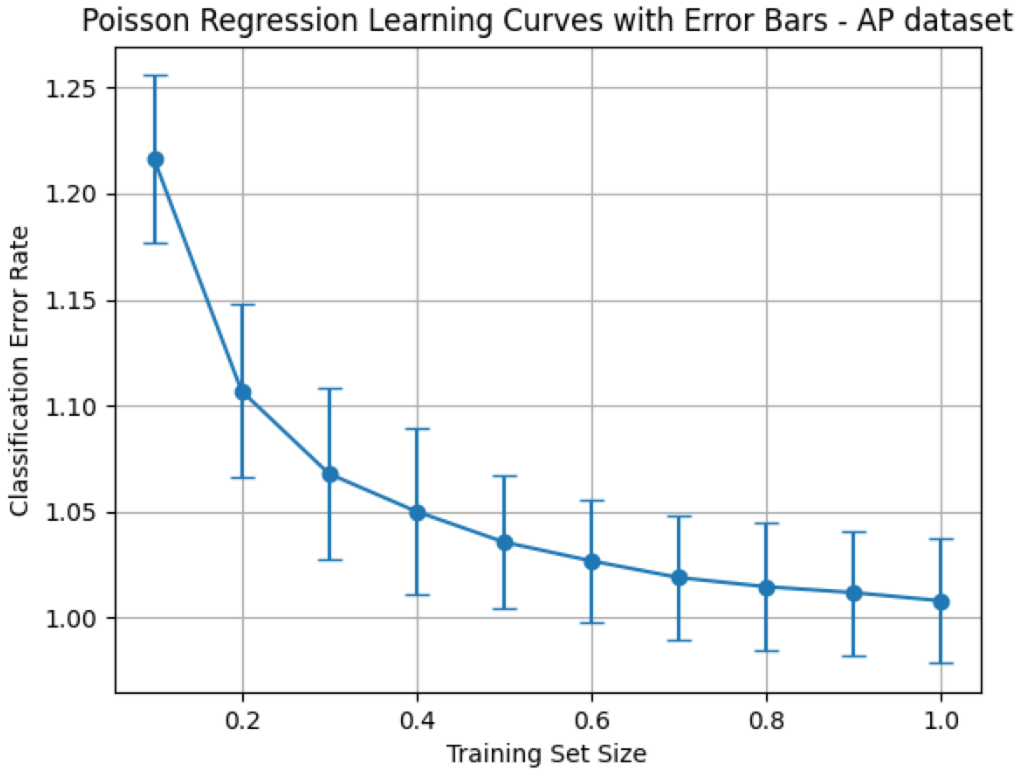




Dataset	# of Features	# Labels	*Runtime*	*Avg Iterations*
A	60	2000	0.26459	2.0267
usps	256	1540	2.81839	5.52333

Observations:
Both learning curves for A and usps datasets tend to converge on its overall lowest classification error rate with the largest training set set size. The learning curve for Dataset A shows a relatively quick convergence with a pretty significant drop in classification error between training set sizes 0.0 and 0.4. However, usps is seen to experience a minor uptick around the 0.5 training set size after which it continues the trend of a dropping classification error rate, indicative of a slower but steady convergence. One can also notice that usps takes almost twice the number of iterations and significantly more runtime, this can likely be attributed to there being a much higher number of features in ups (256) as opposed to A's (60).

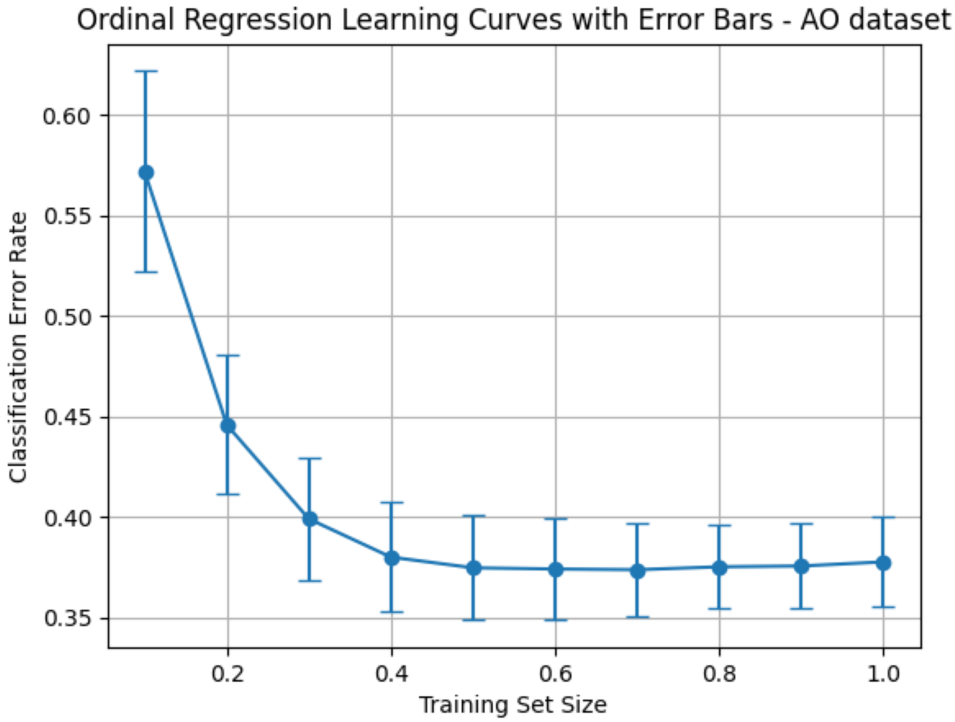
Poisson Regression:



Dataset	Features	Labels	*Runtime*	*Avg Iterations*
AP	60	2000	0.5523	6.73

Observations:
The learning curve for Poisson regression on Dataset AP shows slower convergence and higher error rates when compared to the previously seen logistic regression as seen by the range of error rates falling between [1, 1.25] vs logistic regression's significantly lower ranges. The error rates once again decrease slightly as the training set size increases reaching its lowest classification error rate with the largest training size. Moreover, when compared to dataset A's logistic regression error curve, one can see that the highest drop on error also is predominant once again between 0.0 - 0.4 permutations of the training set size (mostly between 0.0 and 0.2). Once again, with a comparable dataset size to that of A's with logistic regression, one can observe that the runtime and avg number of iterations are much higher here than that of A's

Ordinal Regression:



Dataset	Features	Labels	*Runtime*	*Avg Iterations*
AO	60	2000	3.986	5.8033

Observations:

Similar to Logistic Regression on Datasets A and usps and Poisson Regression on Dataset AP, the learning curve for Ordinal Regression on Dataset AO tends to converge towards its lowest classification error rate with the largest training set size. Between training set sizes of approximately 0 and 4, there is a significant and decelerating decreasing trend in error rates. During this range, the model's performance improves substantially, leading to a noticeable drop in classification errors. However, unlike, poisson and logistic regression, can notice that the model doesn't improve much after the 0.4 training set size. Dataset AO also has a comparable dataset size to that of Dataset A but with logistic regression. Despite this, the runtime and average number of iterations for Ordinal Regression are much higher. This can be attributed to the inherent complexity of the ordinal regression model, which requires more extensive computations and iterations to achieve convergence.

Codebase:

```
import numpy as np
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import pandas as pd
import time


def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def glm(X, y, glm_type, alpha=10, max_iterations=100, tol=1e-3):
    m, n = X.shape
    w = np.zeros(n)  # Initialize weight vector
    num_iterations = 0

    for iteration in range(max_iterations):
        if glm_type == "Logistic":
            y_pred = sigmoid(X.dot(w))
            d = y - y_pred
            R = y_pred * (1 - y_pred)
        elif glm_type == "Poisson":
            y_pred = np.exp(X.dot(w))
            d = y - y_pred
            R = y_pred
        elif glm_type == "Ordinal":
            s = 1.5  # Ordinal regression parameter s
            phi = [-np.inf, -2, -1, 0, 1, np.inf]  # Ordinal regression thresholds
            K = len(phi)
            ai = X.dot(w)
            yi = np.zeros((m, K))

            for j in range(K):
                yi[:, j] = sigmoid(s * (phi[j] - ai))

            # Compute di and ri for each example i with label ti
            d = np.zeros(m)
            R = np.zeros(m)
            for i in range(m):
                ti = int(y[i])  # The ordinal label
                d[i] = yi[i, ti] + yi[i, ti - 1] - 1
                R[i] = (s ** 2) * ((yi[i, ti] * (1 - yi[i, ti])) + (yi[i, ti - 1] * (1 - yi[i, ti - 1])))

        H = -X.T.dot(R[:, np.newaxis] * X) - (alpha * np.eye(n))
        g = X.T.dot(d) - (alpha * w)
```

```
        # Update weights using Newton's method
        w_new = w - np.linalg.solve(H, g)

        # Check for convergence
        if np.linalg.norm(w_new - w) < tol:
            break

        w = w_new
        num_iterations += 1

    return w, num_iterations


def glm_output_run():
    files = {"Logistic": ["A", "usps"], "Poisson": ["AP"], "Ordinal": ["AO"]}

    for glm_type, datasets in files.items():
        for d in datasets:
            data = pd.read_csv('pp3data/' + d + '.csv', header=None)
            labels = pd.read_csv('pp3data/labels-' + d + '.csv', header=None)

            X = data.values
            y = labels.values.flatten()
            print("X-shape: " +str(X.shape))
            print("Y-shape: "+str(y.shape))
            runs = 30
            training_set_sizes = np.linspace(0.1, 1.0, 10)
            error_rates = np.zeros((runs, len(training_set_sizes)))
            iterations = []
            start_time = time.time()

            for run in range(runs):
                np.random.seed(run)
                permuted_indices = np.random.permutation(X.shape[0])
                test_size = X.shape[0] // 3
                test_indices = permuted_indices[:test_size]
                train_indices = permuted_indices[test_size:]

                X_train, y_train = X[train_indices], y[train_indices]
                X_test, y_test = X[test_indices], y[test_indices]

                # Initialize arrays to store error rates for different training set sizes
                run_error_rates = np.zeros(len(training_set_sizes))

                for i, train_size in enumerate(training_set_sizes):
                    train_size = int(train_size * len(X_train))
                    X_train_subset, y_train_subset = X_train[:train_size], y_train[:train_size]
```

```
            # Train the logistic regression model
            w_MAP, num_iterations = glm(X_train_subset, y_train_subset, glm_type)
            iterations.append(num_iterations)

            error = 0
            if glm_type == "Logistic":
                y_pred = sigmoid(X_test.dot(w_MAP))
                error = (y_pred >= 0.5) != y_test

            elif glm_type == "Poisson":
                y_pred = np.exp(X_test.dot(w_MAP))
                error = np.abs(y_pred - y_test)

            elif glm_type == "Ordinal":
                phi = [-np.inf, -2, -1, 0, 1, np.inf]
                K = len(phi)
                s = 1.5
                a = X_test.dot(w_MAP.T)
                yj = np.zeros((X_test.shape[0], K))

                for j in range(K):
                    yj[:, j] = sigmoid(s * (phi[j] - a))

                pj = np.zeros((X_test.shape[0], K))
                pj[:, 0] = yj[:, 0]
                for j in range(1, K):
                    pj[:, j] = yj[:, j] - yj[:, j - 1]

                # Predict the ordinal labels t^
                predicted_labels = np.argmax(pj, axis=1)

                # Calculate the absolute error err
                true_labels = y_test
                error = np.abs(predicted_labels - true_labels)

                # Calculate the average error rate
            run_error_rates[i] = np.mean(error)

        error_rates[run] = run_error_rates

    end_time = time.time()
    # Calculate mean and standard deviation of error rates for each training set size
    mean_error_rates = np.mean(error_rates, axis=0)
    print("Mean error rates: "+str(mean_error_rates))
    std_dev_error_rates = np.std(error_rates, axis=0)
```

```
            print(glm_type + " Regression - " + d + " dataset")
            print("Average Iterations: "+str(np.mean(iterations)))
            print("Runtime: "+str(end_time - start_time))
            print("\n")

            # Plot the learning curves with error bars
            plt.errorbar(training_set_sizes, mean_error_rates,
yerr=std_dev_error_rates, fmt='-o', capsize=5)
            plt.xlabel("Training Set Size")
            plt.ylabel("Classification Error Rate")
            plt.title(glm_type + " Regression Learning Curves with Error Bars -
" + d + " dataset")
            plt.grid(True)
            plt.show()

glm_output_run()
```

README:

```
Just run the PP3.py file to get the 4 plots. You're gonna have to close the
plot window each time for the next plot to appear. The terminal should also
show the
X-shape, y-shape, Mean error rates, the dataset we are looking at, the average
number of iterations and the runtime per dataset.
```