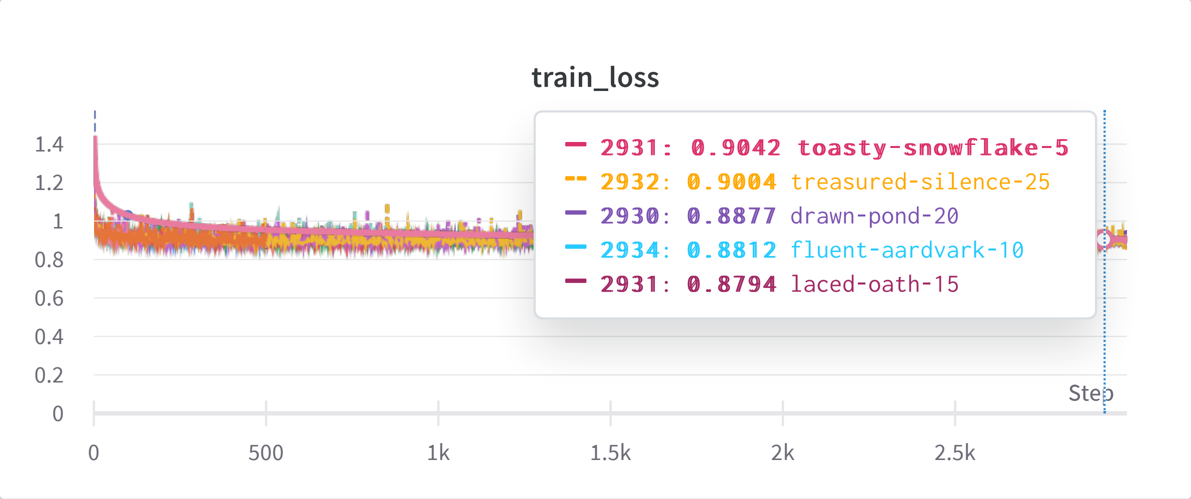
Share Comment Star

Multinomial Logistic Regression

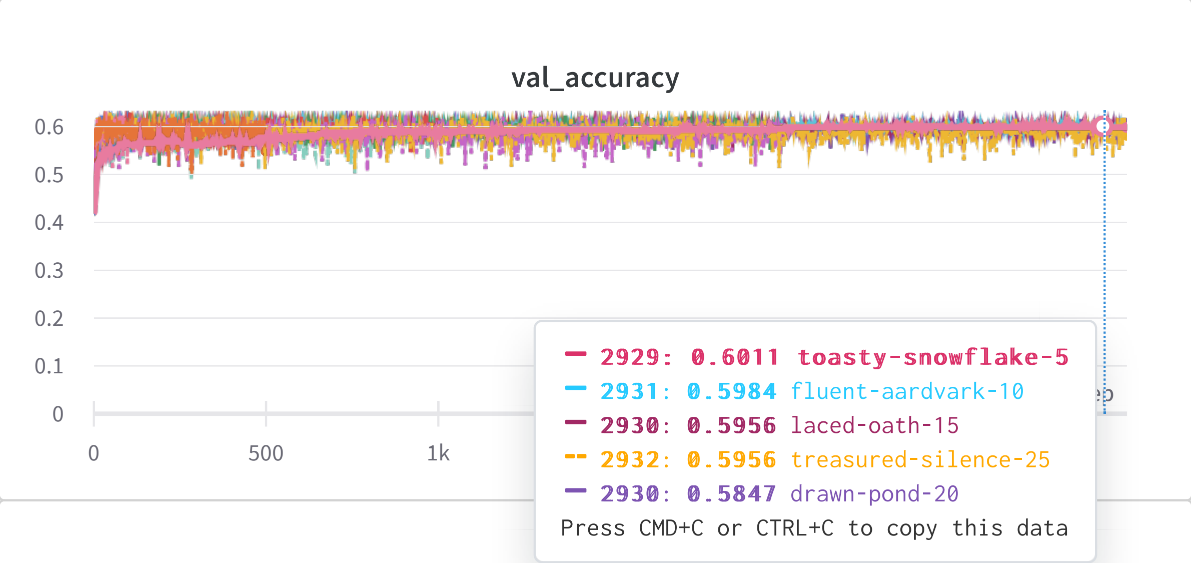
SMAI - Assignment 3

Ashna Dua

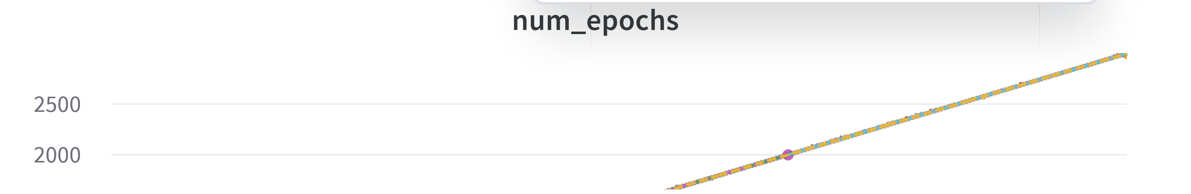
Plots



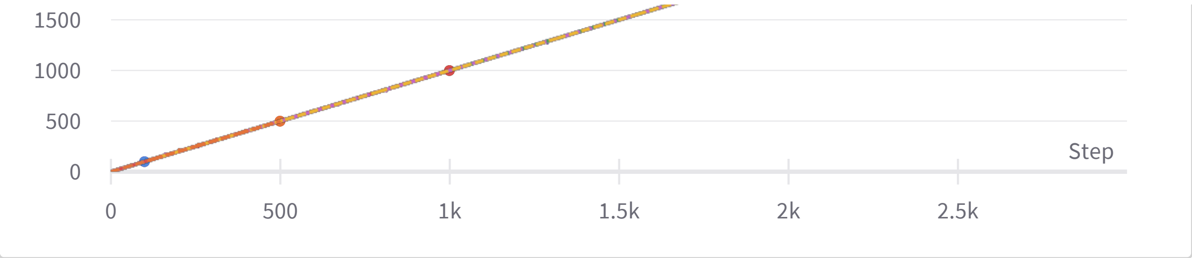
train_loss

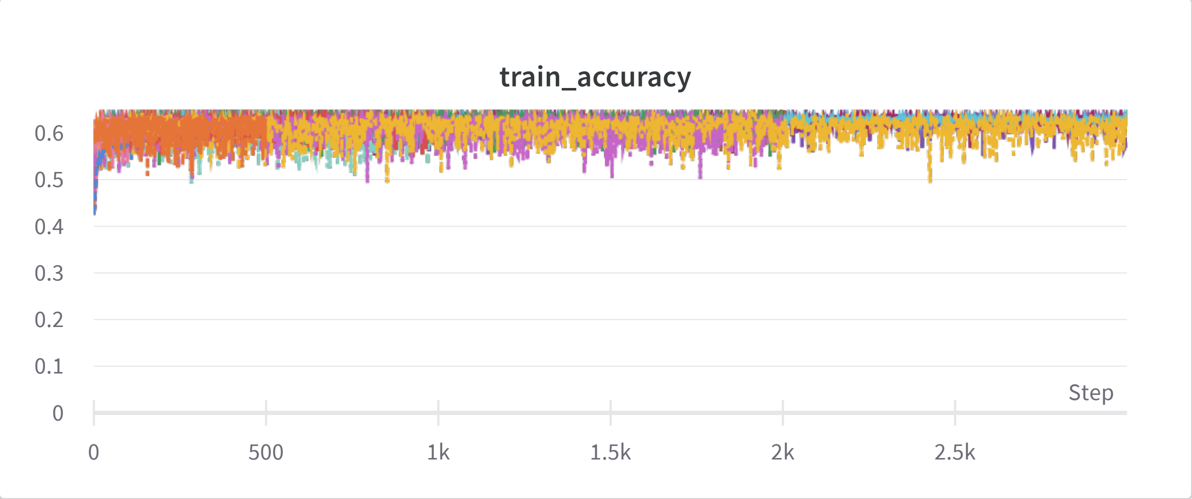


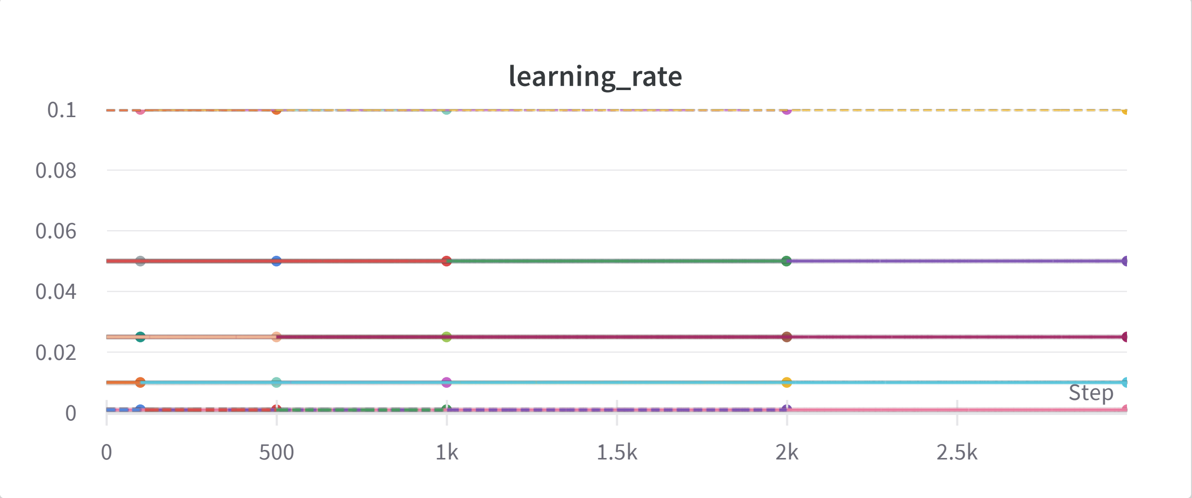
val_accuracy

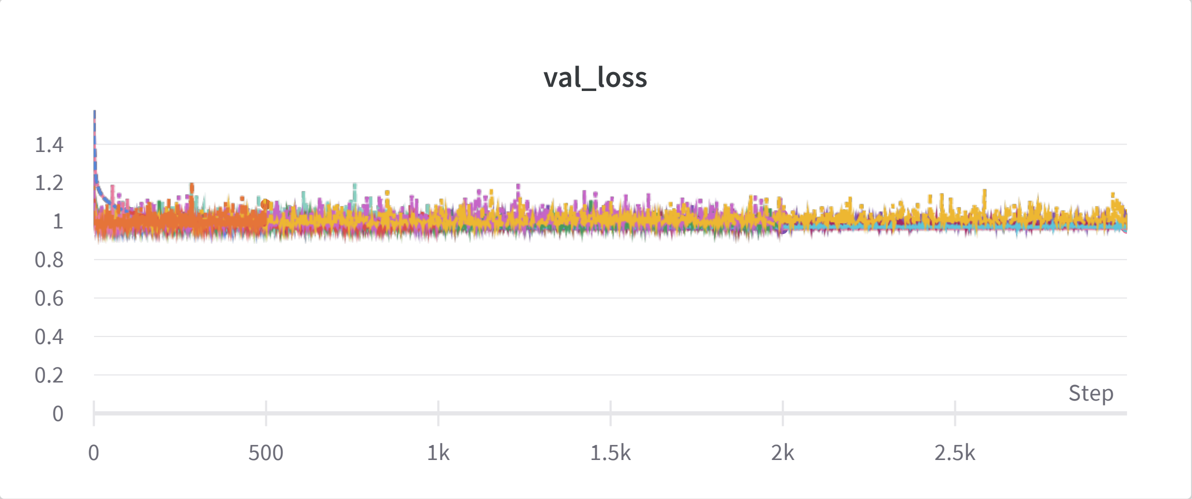


num_epochs







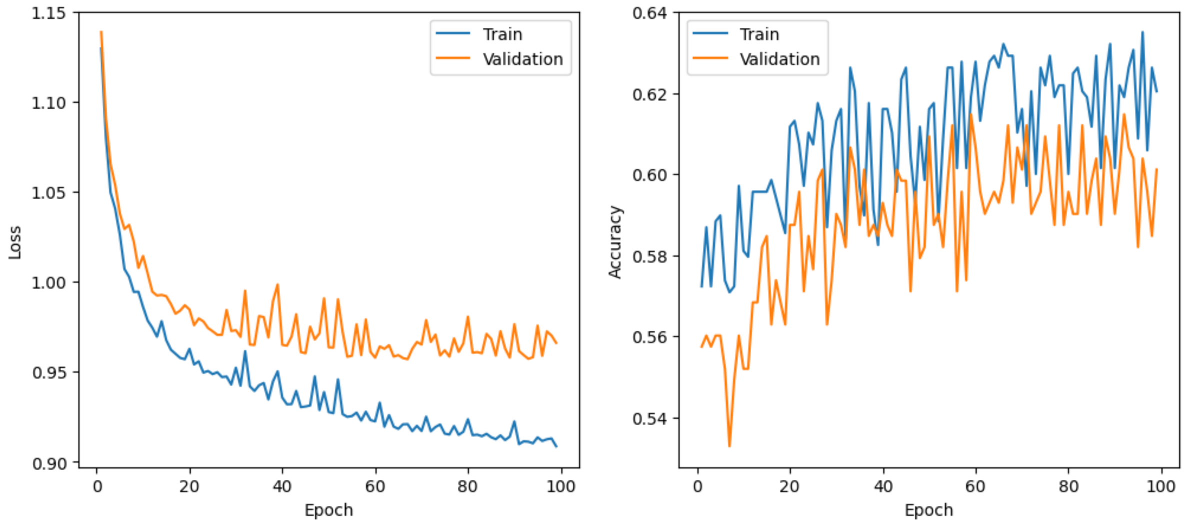


In Multinomial Logistic Regression, 5 configurations of learning rates, i.e. 0.001, 0.01, 0.025, 0.05, and 0.1 and 5 configurations of epochs, i.e. 100, 500, 1000, 2000, and 3000. Training loss, accuracy and Validation loss, accuracy were logged for each epoch, and the best hyper-parameters achieved were:

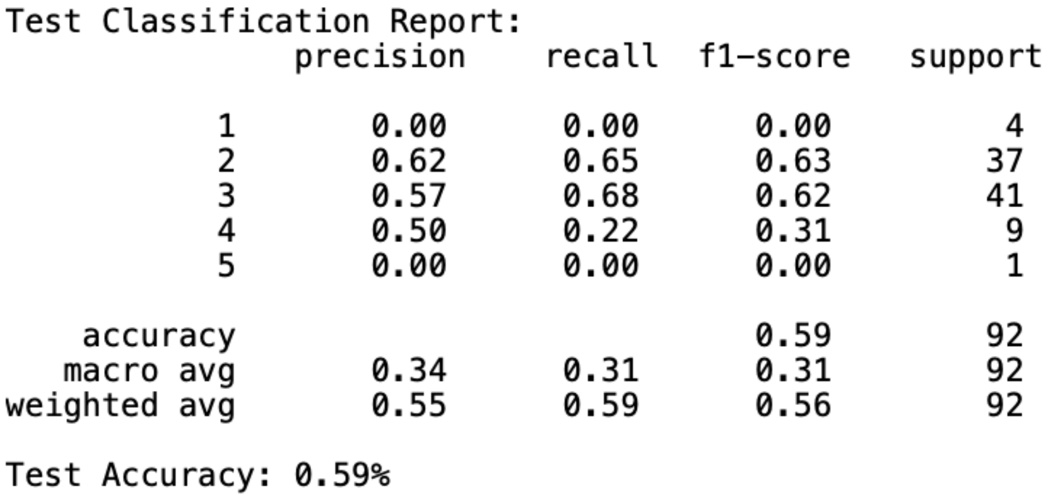
- Train Loss: 0.9277
- Train Accuracy: 0.62%
- Validation Loss: 0.9636
- Validation Accuracy: 0.61%

This model is considered as the best performing model for Multinomial Logistic Regression with 99 epochs and 0.025 learning rate.

```
class MultinomialLogisticRegression:
    def __init__(self, learning_rate=0.1, num_epochs=1000, log_flag=
        self.learning_rate = learning_rate
        self.num_epochs = num_epochs
        self.log_flag = log_flag

    def softmax(self, z):
        exp_z = np.exp(z - np.max(z, axis=1, keepdims=True))
        return exp_z / np.sum(exp_z, axis=1, keepdims=True)

    def cross_entropy_loss(self, y_true, y_pred):
        epsilon = 1e-10
        num_samples = y_true.shape[0]
        y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
        loss = -np.sum(y_true * np.log(y_pred)) / num_samples
        return loss

    def gradient_descent(self, X, y, y_pred):
        num_samples = X.shape[0]
        grad = np.dot(X.T, (y_pred - y)) / num_samples
        return grad

    def plot(self, train_losses, val_losses, train_accuracies, val_a
        epochs = range(1, self.num_epochs + 1)
        plt.figure(figsize=(12, 5))
        plt.subplot(1, 2, 1)
        plt.plot(epochs, train_losses, label='Train')
        plt.plot(epochs, val_losses, label='Validation')
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.legend()

        plt.subplot(1, 2, 2)
        plt.plot(epochs, train_accuracies, label='Train')
        plt.plot(epochs, val_accuracies, label='Validation')
        plt.xlabel('Epoch')
        plt.ylabel('Accuracy')
        plt.legend()
        plt.show()

    def train(self, X, y, X_val, y_val):
        num_samples, num_features = X.shape
        num_classes = y.shape[1]
        self.weights = np.zeros((num_features, num_classes))

        train_losses = []
        val_losses = []
        train_accuracies = []
        val_accuracies = []

        for epoch in range(self.num_epochs):
            indices = np.random.permutation(num_samples)
            X_shuffled = X[indices]
            y_shuffled = y[indices]

            for i in range(num_samples):
                xi = X_shuffled[i:i+1]
                yi = y_shuffled[i:i+1]
                z = np.dot(xi, self.weights)
                y_pred = self.softmax(z)
                gradient = self.gradient_descent(xi, yi, y_pred)
                self.weights -= self.learning_rate * gradient
            y_train_pred = self.predict(X)
            y_train_pred_labels = np.argmax(y_train_pred, axis=1)
            train_loss = self.cross_entropy_loss(y, y_train_pred)
            train_losses.append(train_loss)
            train_accuracy = accuracy_score(np.argmax(y, axis=1), y_
            train_accuracies.append(train_accuracy)

            y_val_pred = self.predict(X_val)
            y_val_pred_labels = np.argmax(y_val_pred, axis=1)
            val_loss = self.cross_entropy_loss(y_val, y_val_pred)
            val_losses.append(val_loss)
            val_accuracy = accuracy_score(np.argmax(y_val, axis=1),
            val_accuracies.append(val_accuracy)

            if (epoch + 1) % 50 == 0:
                print(f"Epoch {epoch + 1}/{self.num_epochs}")
                print(f"Train Loss: {train_loss:.4f}")
                print(f"Train Accuracy: {train_accuracy:.2f}%")
                print(f"Validation Loss: {val_loss:.4f}")
                print(f"Validation Accuracy: {val_accuracy:.2f}%\n"
                classification_rep = classification_report(np.argma
                print("Validation Classification Report:\n", classif

            if self.log_flag == 1:
                wandb.log({
                    "learning_rate": self.learning_rate,
                    "num_epochs": epoch,
                    "train_loss": train_loss,
                    "val_loss": val_loss,
                    "train_accuracy": train_accuracy,
                    "val_accuracy": val_accuracy
                })

        self.train_losses = train_losses
        self.val_losses = val_losses
        self.train_accuracies = train_accuracies
        self.val_accuracies = val_accuracies
        self.plot(train_losses, val_losses, train_accuracies, val_ac

    def predict(self, X):
        z = np.dot(X, self.weights)
        return self.softmax(z)
```



The Classification Report for the hyper-parameters is as follows:



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

           1       0.00      0.00      0.00         4
           2       0.62      0.65      0.63        37
           3       0.57      0.68      0.62        41
           4       0.50      0.22      0.31         9
           5       0.00      0.00      0.00         1

    accuracy                           0.59        92
   macro avg       0.34      0.31      0.31        92
weighted avg       0.55      0.59      0.56        92

Test Accuracy: 0.59%
```

Created with ❤️ on Weights & Biases.

https://wandb.ai/ashna-dua/MultinomialLogisticRegression/reports/Multinomial-Logistic-Regression--Vmlldzo1NzUyNTUw