CS 412 Homework 3

Şevki Aybars Türel - 28238

27.04.2024

1 Load the Dataset and Preprocess the Data

1.1 Loading Dataset

Pandas library was used to open the given csv. Basic data analysis was done on the given dataset. Number of features and number of instances was detected by using shape and head functions. Similarly unique values in the given features are searched by using unique function.

1.2 Splitting Data

Dataset must be split into training, validation and test sets. Firstly, 60% of the all data split and it is for training. After 40% of the all data split into two equal dataset which are for test and validation. As a result, there are 357 instances for training. Additionally, 178 instances for validation and 179 for test. Random state is arranged to 42.

```
training_data, test_and_validation_data = train_test_split(titanic_data, test_size=0.4, random_state=42)

validation_data, test_data = train_test_split(test_and_validation_data, test_size=0.5, random_state=42)
```

Figure 1: Code line developed for splitting data.

1.3 Scaling Data

In homework documentation, it is mentioned that StandartScaler function can be used for scaling purposes. However, in documentation features, must be between 0-1. When I used StandartScaler our feature values can be negative. Therefore I used MinMaxScaler function to satisfy the 0-1 values in the feature. Training dataset scaled by using fit_transform function because it will arrange all feature according to given dataset. For the validation and test dataset, I used transform function because these datasets must be scaled according to max and min value of the training dataset to prevent data leakage.

```
scaler = MinMaxScaler()

training_data_scaled = pd.DataFrame(scaler.fit_transform(training_data), columns=training_data.columns)

validation_data_scaled = pd.DataFrame(scaler.transform(validation_data), columns=validation_data.columns)

test_data_scaled = pd.DataFrame(scaler.transform(test_data), columns=test_data.columns)
```

Figure 2: Code line developed for scaling data.

1.4 Splitting Features and Labels

Features and labels must be arranged because it will make easier to training model process. Features include PClass, Sex and Age columns while labels include only Survived column.

```
X_training_data = training_data_scaled.drop('Survived', axis=1)
y_training_data = training_data_scaled['Survived']
X_validation_data = validation_data_scaled.drop('Survived', axis=1)
y_validation_data = validation_data_scaled['Survived']
X_test_data = test_data_scaled.drop('Survived', axis=1)
y_test_data = test_data_scaled['Survived']
```

Figure 3: Code line developed for splitting label and features.

1.5 Adding Bias to Dataframes

I need to add new column named as Bias all the values are equal to 1 because weights has bias therefore it has four different value in weights. If all the values must be multiplied by the features, I need to add one more feature named as Bias. I used np.hstack because I want to return them to array object for training purposes.

```
features_train_biased = np.hstack([np.ones((X_training_data.shape[0], 1)), X_training_data.values])
features_val_biased = np.hstack([np.ones((X_validation_data.shape[0], 1)), X_validation_data.values])
features_test_biased = np.hstack([np.ones((X_test_data.shape[0], 1)), X_test_data.values])

features_train_biased
```

```
array([[1.        , 0.5       , 1.        , 0.77381252],
       [1.        , 1.        , 1.        , 0.09525006],
       [1.        , 1.        , 1.        , 0.39683338],
       ...,
       [1.        , 1.        , 1.        , 0.50992712],
       [1.        , 0.        , 0.        , 0.17064589],
       [1.        , 0.        , 1.        , 0.25860769]])
```

Figure 4: Code line developed for adding bias.

2 Implementing the Logistic Regression Model

2.1 Weight Generation Function

Weight function generates random numbers between 0 and 1. It will generate four numbers and store it in the vector. Therefore, it will generate starting weight vector. It is dynamic function. If you change your dataset, it will generate feature + 1 weights.

```
def weight_generation(features):
    return np.random.rand(features.shape[1])
```

Figure 5: Code line developed for weight generation.

2.2 Sigmoid Function

It is basic mathematic function.

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

Figure 6: Code line developed for sigmoid function.

2.3 Implementing Cost Function

It is basic mathematic function.

```
def logistic_cost_function(X, y, weights):
    m = len(y)
    predictions = sigmoid(np.dot(X, weights))
    cost = -(1/m) * (np.sum(y * np.log(predictions) + (1 - y) * np.log(1 - predictions)))
    return cost
```

Figure 7: Code line developed for cost function.

2.4 Implementing Gradient Descent Algorithm

It is the mathematical function that is taken from homework file. It will store the training cost and validation code separately.

```python
def gradient_descent(X_training_data, y_training_data, X_validation_data, y_validation_data, weights, alpha, num_iterations):
    training_cost_history = []
    validation_cost_history = []

    for i in range(num_iterations):
        predictions_train = sigmoid(np.dot(X_training_data, weights))
        error_train = predictions_train - y_training_data
        gradient = np.dot(X_training_data.T, error_train) / len(y_training_data)
        weights -= alpha * gradient
        train_cost = logistic_cost_function(X_training_data, y_training_data, weights)
        val_cost = logistic_cost_function(X_validation_data, y_validation_data, weights)

        training_cost_history.append(train_cost)
        validation_cost_history.append(val_cost)

    return weights, training_cost_history, validation_cost_history
```

Figure 8: Code line developed for Gradient Descent.

3 Training Model

3.1 Model Training

Learning rate is decided as 0.1 and iteration is 100.

```python
weight = weight_generation(features_train_biased)

learning_rate = 0.1
iterations = 100

final_weights_biased, training_losses_biased, validation_losses_biased = gradient_descent(
    features_train_biased, y_training_data, features_val_biased, y_validation_data, weight, learning_rate, iterations)
```

Figure 9: Code line developed for training model.

3.2 Loss Visualization

Loss for validation and training can be observed in the following graph. It can change because weight produces random weight. Every iteration it can change. When the iteration is increasing the loss is decreasing.

```python
weight = weight_generation(features_train_biased)

learning_rate = 0.1
iterations = 100

final_weights_biased, training_losses_biased, validation_losses_biased = gradient_descent(
    features_train_biased, y_training_data, features_val_biased, y_validation_data, weight, learning_rate, iterations)
```

Figure 10: Validation and Training Loss Graph.

4 Varying Step Size for Best Parameter

4.1 Iteration Numbers Alpha Sizes

Various step sizes will be iterated with various numbers. You can see the numbers below.

```
step_sizes = [0.01, 0.05, 0.07, 0.1, 0.3 , 0.5, 0.7, 0.9, 1]
num_iterations = [50, 100, 150, 200, 250]

best_params, all_losses = hyperparameter_decider(features_train_biased, y_training_data, features_val_biased, y_validation_data, step_sizes, num_iterations)
```

Figure 11: Alpha and iteration numbers.

4.2 Best Alpha and Iteration Number

It can be changed because of the random weights. However most time Alpha is 0.9 or 1 and iteration number is 250

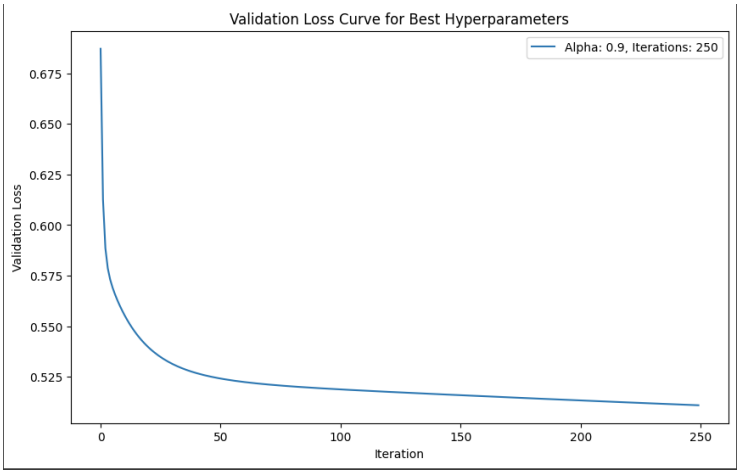

Figure 12: Best Hyper parameters.

5 Combining Validation and Training Data

Training data and validation test will be concatenated and it creates o dataset that includes both training data and validation data. After that, the new model was trained by using the best hyperparamters. In this case alpha = 0.9 and iteration is 250. It has the smallest loss value. It can change for every iteration.

6 Evaluating Model

The accuracy of the new model is 79.33%. It can also change because of the random weight. I put 0.5 as a decision threshold. If decision value higher than 0.5 its label is 1 and if it is smaller than 0.5 it is labeled with 0. After that, I compared guessed label and the true label and take the proportions of the true label to all labels.