Task 1: Logistic Regression Classifier from Scratch

Nghia Nim

Tasnim Ahmed

Helin Mazi

Ameena Zewail

New York University Abu Dhabi

29 February 2024

A. Introduction:

Logistic Regression is a widely used statistical method used for classification and prediction analysis. It is a fundamental tool in the field of machine learning originally developed for binary classification problems. Logistic regression models have evolved to be applicable across various domains due to their simplicity, interpretability, and efficiency.

Logistic Regression models the relationship between the independent variable(s) and the dependent variable to be predicted. The outcome of the model is a probability which means it is a value between 0 and 1. To implement a logistic regression model, a log likelihood expression is written which helps find the best weight parameters that can be used to predict the most accurate probabilities of outcomes.

While various machine learning libraries offer pre-built Logistic Regression models, understanding the inner workings of the algorithm can be insightful. This includes understanding the logistic function, or the sigmoid function, which transforms the linear combination of input features into probabilities or values between 0 and 1.

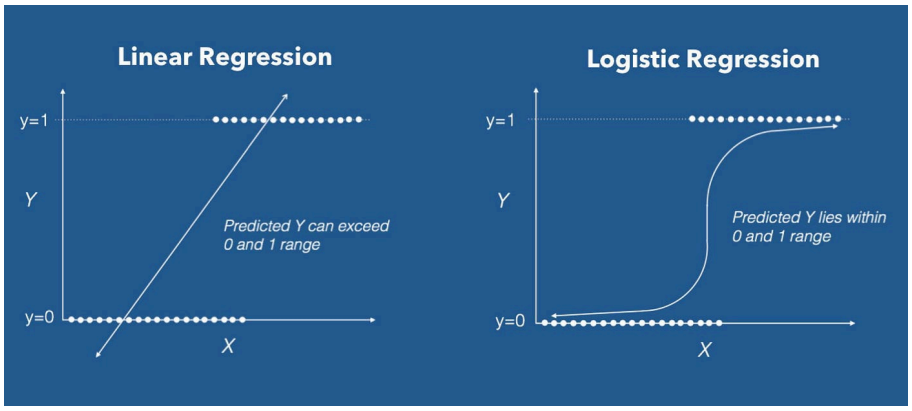

Figure 1. The sigmoid function compared to a linear function

The mathematical foundation of Logistic Regression involves the likelihood function, a key component in maximum likelihood estimation. By maximizing the likelihood function, Logistic Regression determines the optimal parameters that define the decision boundary between classes as previously mentioned.

The log-likelihood function for logistic regression can be written as:

$$\text{Log likelihood} = \sum[y_i ln\hat{y}_i + (1 - y_i)\ln(1 - \hat{y}_i)]$$

The negative log-likelihood function is the function we aim to minimize during the training of the logistic regression model. To check if the model predicted correctly, we aim to minimize this function to make predicted outputs closer to real/true value

$$J(\mathbf{w}, b) = -\frac{1}{N} \sum_{i=1}^{N} \left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right]$$

Gradient descent is an iterative optimization algorithm used to find the minimum value of the negative likelihood function. In the context of logistic regression, the gradients of the loss function with respect to the weights and bias are computed. The weights and bias are then updated in the opposite direction of the gradients to minimize the loss.

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_k} = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{\exp\left(\boldsymbol{w}_k^\top \boldsymbol{x}_i\right)}{\sum_{\ell=1}^{K} \exp\left(\boldsymbol{w}_\ell^\top \boldsymbol{x}_i\right)} - y_{ik} \right) \boldsymbol{x}_i$$

In the subsequent sections, we will navigate through the steps of implementing a Logistic Regression classifier from scratch, providing code snippets, data visualization and accuracy calculation.

B. Objective:

Our goal for this project is to Implement and train a logistic regression classifier to solve a binary classification problem without using high-level machine learning libraries. The model should be able to predict whether imputed features for a flower can classify it as a Setosa or not. This goal can be achieved by minimizing the loss function to produce the highest accuracy in the prediction of the model.

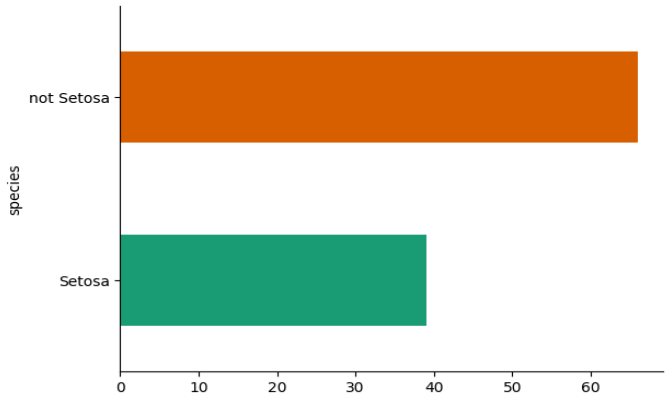

Figure 2. The frequency of Setosa (1) and not Setose (0) in training data

C. Methodology:

To implement an accurate predicting model simple machine learning libraries such as Python, NumPy, Matplotlib, and Pandas were used. These aid with data manipulation and visualization.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # data plotting
import os
```

The provided data was then stored into a pandas DataFrame object called data.

```
#importing the data
data = pd.read_csv('data_train.csv')
```

The data has 105 elements to facilitate supervised learning. Each data point has 5 features in which the 5th one is the actual classification. Below are 5 sample elements from the data set:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.7	3.8	1.7	0.3	Setosa
1	6.7	2.5	5.8	1.8	not Setosa
2	5.1	2.5	3.0	1.1	not Setosa
3	4.8	3.0	1.4	0.3	Setosa
4	6.2	2.2	4.5	1.5	not Setosa

The following graph helps with data visualization, a feature of the `matplotlib.pyplot` library:

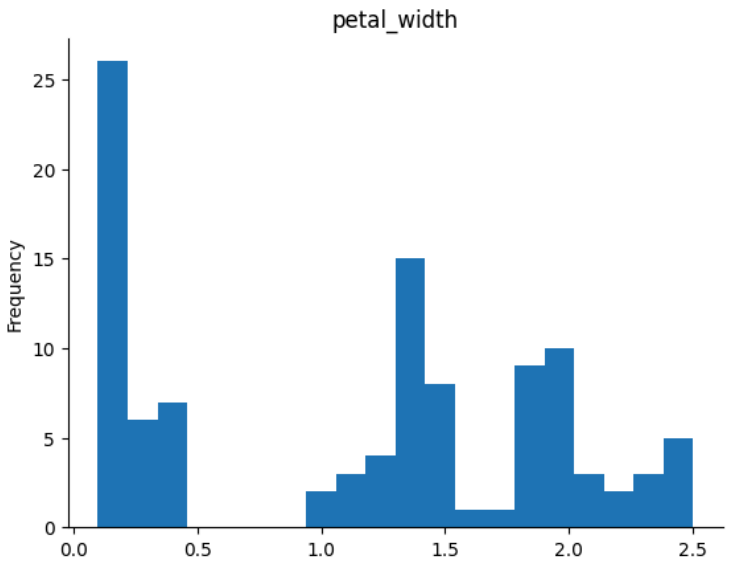

Figure 3. Histogram of frequencies of different petal_width values in training data

Data visualization assisted us in assessing the results of the model. The graph above illustrates the petal width frequencies. A notable spike in frequency occurs for petal widths between 0 and 0.2. Moreover, the graph indicates a lack of petal width data between 0.5 and 0.9, providing insights on the values possibly to be used to classify the test data.

D. Code Implementation:

```
import pandas as pd
from sklearn.model_selection import train_test_split

def split_data(df, target_column):
    X = df.drop(target_column, axis=1)  # Drop the target column to get the features
    y = df[target_column]               # Get only the target column
    return X, y

X, y = split_data(data, 'species')

y_binary = (y == 'Setosa').astype(int)

X_train, X_test, y_train, y_test = train_test_split(X, y_binary, test_size=0.25, random_state=42)
```

Firstly, the code imports the necessary libraries: Pandas for data manipulation and scikit-learn's train_test_split function for splitting the dataset into training and testing sets. The split_data function defined takes a DataFrame (df) and a target column named (target_column) as inputs and splits data into features (X) and target (y). Here (X) represents the features or the independent variables of the dataset. It is obtained by dropping the target column (species) from the original DataFrame data. On the other hand, (y) represents the target variable (dependent variable) of the dataset which takes values from the original 'species' column of the DataFrame data. Then, a binary target variable y_binary is created, which is 1 if the original target column

(species) is 'Setosa' and 0 otherwise. Finally, train_test_split is used to split the features (X) and the binary target (y_binary) into training and testing sets (X_train, X_test, y_train, y_test) with a test size of 25% and a fixed random state for reproducibility. This prepares the data for training and evaluating a binary classification model.

```
import numpy as np

class Logistic_Regression:
    def __init__(self):
        self.weights = None
        self.bias = None

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

    def fit(self, X, y, learning_rate=0.01, n_iterations=1000):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(n_iterations):
            model = np.dot(X, self.weights) + self.bias
            predictions = self.sigmoid(model)

            dw = np.dot(X.T, (y - predictions)) / n_samples
            db = np.sum(y - predictions) / n_samples

            self.weights += learning_rate * dw
            self.bias += learning_rate * db

    def predict_prob(self, X):
        model = np.dot(X, self.weights) + self.bias
        return self.sigmoid(model)

    def predict(self, X, threshold=0.5):
        probabilities = self.predict_prob(X)
        return np.where(probabilities >= threshold, 1, 0)



model = Logistic_Regression()
model.fit(X_train, y_train)
```

The code imports the NumPy library as np for numerical computations. The Logistic_Regression class is initialized with weights and bias set to None. The methods for sigmoid, fit, predict_prob and predict are then defined. The sigmoid method computes the sigmoid function, which is used to squash the model output to a range between 0 and 1, thereby representing probabilities. The fit method trains the logistic regression model using gradient descent optimization. It iterates through a fixed number of iterations (n_iterations) to update the weights and bias by computing gradients and adjusting them with the provided learning rate. The predict_prob method predicts probabilities of class membership for given input features X. Finally, the predict method makes binary predictions based on the predicted probabilities, using

a threshold value of 0.5 to classify the samples into two classes. An instance of the Logistic_Regression class is then created as a model, and it is trained on the training data (X_train, y_train) using the fit method. This instance can now be used to make predictions on new data.

E. Validation and Testing:

```
from sklearn.metrics import accuracy_score
model = Logistic_Regression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
```

```
Accuracy: 1.0000
```

This code snippet focuses on evaluating the performance of the trained logistic regression model using validation data and testing data. It imports the accuracy_score function from the sklearn.metrics module, which will be used to compute the accuracy of the model's predictions. After initializing an instance of the Logistic_Regression class and fitting it to the training data (X_train, y_train), the model is used to predict the target variable (y_pred) for the testing features (X_test). Then, the accuracy_score function compares the predicted target values (y_pred) with the actual target values from the testing data (y_test) and computes the accuracy of the model. The accuracy is a measure of the proportion of correctly classified samples. Finally, the computed accuracy score is printed to the console with four decimal places for precision. This accuracy score provides insight into how well the logistic regression model performs on unseen data, helping to evaluate its generalization ability.

F. Discussion:

Achieving a perfect accuracy score of 1.0000 on the testing data underscores the effectiveness of the logistic regression model in accurately classifying the Iris dataset. This outcome highlights several strengths of the implementation, including the model's capability to effectively separate the classes within the dataset and its simplicity, making it easy to understand and implement. However, it is crucial to recognize the limitations inherent in this scenario. The Iris dataset is relatively small, containing only three classes and four features, which might have contributed to the high accuracy. This realization prompts a consideration of potential challenges in scaling the model to larger and more complex datasets, where performance might not be as optimal.

Throughout the process, valuable insights were gained into the interpretation of logistic regression models and the impact of hyperparameters on model convergence and performance. Working with logistic regression provided a deeper understanding of how the model makes

predictions based on linear combinations of features and how the sigmoid activation function transforms these predictions into probabilities. Challenges such as ensuring model convergence and interpreting evaluation metrics were addressed through careful experimentation with hyperparameters and critical analysis of model performance.

Moving forward, it is essential to consider alternative evaluation metrics beyond accuracy, such as precision, recall, and F1-score, to gain a more comprehensive understanding of the model's performance, especially in scenarios with class imbalances or more complex datasets. Reflecting on the strengths, limitations, and insights gained from this implementation will inform future iterations and improvements, guiding the development of more robust and generalizable machine learning models.

G. Conclusion:

In summary, the implementation of logistic regression on the Iris dataset resulted in a remarkable achievement of perfect accuracy on the testing data. While this success highlights the model's simplicity and effectiveness, it also raises considerations regarding potential overfitting due to the dataset's simplicity and small size. Nonetheless, understanding and implementing logistic regression from scratch provided valuable insights into its workings, emphasizing the significance of interpretability and the need for careful hyperparameter tuning. Overall, this exercise underscores logistic regression's potential as a powerful tool for classification tasks and emphasizes the importance of critical evaluation and continuous improvement in developing robust machine-learning models.

H. References:

Pant, A. (2021, December 7). Introduction to Logistic Regression - towards data science. Medium. https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148

What is Logistic regression? | IBM. (n.d.). https://www.ibm.com/topics/logistic-regression#:~:text=Logistic%20regression%20estimates%20the%20probability,given%20dataset%20of%20independent%20variables.