Weronika Jopek, 912400002

1. Logistic Regression

Logistic Regression is a supervised ML algorithm used for classification problems, where the goal is to predict the probability that an instance belongs to a given class. It is commonly used for binary classification, where the sigmoid function is applied.

The sigmoid function is a mathematical function used to map predicted values to probabilities. It transforms any real value into a value between 0 and 1, which is particularly useful for classification tasks. In logistic regression, the model transforms the continuous output from a linear regression function into categorical outputs using the sigmoid function. It takes inputs (independent variables) and outputs a probability value between 0 and 1, making it suitable for binary decisions.

Loss functions are integral to training machine learning models, providing a measure of how well the model's predictions align with actual outcomes. By minimizing the loss, models improve their accuracy. Loss functions can be categorized based on the type of task:

- Regression Models: predict continuous values.

MSE (Mean Squared Error) - it is the Mean of Square of Residuals for all the datapoints in the dataset. Residuals is the difference between the actual and the predicted prediction by the model.

- Classification Models: predict the output from a set of finite categorical values

Cross-Entropy Loss – also known as Negative Log Likelihood. This **loss function** measures how well the predicted probabilities match the actual labels. It increases as the predicted probability diverges from the true label - the farther the model's prediction is from the actual class, the higher the **loss**.

Resluts and Discussion

The obtained results are visible in the plots demonstrating how the learning rate (LR) and the number of iterations affect training accuracy. The first plot shows how LR and iterations affect the training process with MSE loss. A low LR (0.001) leads to slow convergence, with minimal improvement even after many iterations. An LR of 0.01 balances stability and efficiency, yielding higher accuracy as iterations increase. However, an LR of 0.1 causes volatility due to overshooting, especially at higher iterations, indicating the need for careful hyperparameter tuning.

For MSE, accuracy improves gradually over time, especially at lower learning rates. Higher learning rates cause accuracy to fluctuate more, possibly due to instability during model convergence. In contrast, using

Cross-Entropy Loss typically results in faster accuracy increases, particularly with higher learning rates. Cross-Entropy Loss is more sensitive to misclassification of probabilities, allowing the model to adjust more quickly. However, at excessively high learning rates, both MSE and Cross-Entropy exhibit signs of instability, where accuracy plateaus or decreases as the model struggles to converge. The number of iterations plays a crucial role in refining accuracy, with excessive iterations or poor learning rates leading to overfitting or slower convergence.

---

2. Decision Tree

A **decision tree** is a flowchart-like structure used to make decisions or predictions. Structure:

- **Root Node**: represents the entire dataset and the initial decision to be made.
- **Internal Nodes**: represent decisions or tests on attributes. Each internal node has one or more branches.
- **Branches**: represent the outcome of a decision or test, leading to another node.
- **Leaf Nodes**: represent the final decision or prediction. No further splits occur at these nodes.

The process of creating a decision tree involves:

1. **Selecting the Best Attribute**: Using a metric like Gini impurity, entropy, or information gain, the best attribute to split the data is selected.

2. **Splitting the Dataset**: The dataset is split into subsets based on the selected attribute.

3. **Repeating the Process**: The process is repeated recursively for each subset, creating a new internal node or leaf node until a stopping criterion is met (e.g., all instances in a node belong to the same class or a predefined depth is reached).

Metrics for Splitting:

- **Gini Impurity**: Measures the likelihood of an incorrect classification of a new instance if it was randomly classified according to the distribution of classes in the dataset.

$$Gini = 1 - \sum_{i=1}^{n}(p_i)^2,$$

were p_i is the probability of an instance being classifies into a particular class.

Decision trees are easy to understand and interpret. The visual representation closely mirrors human decision-making processes. It can be used for both classification and regression tasks.

Decision trees can easily overfit the training data, especially if they are deep with many nodes. Small variations in the data can result in a completely different tree being generated. Features with more levels can dominate the tree structure.

The depth of a decision tree is the length of the longest path from the root node to a leaf node. A shallow tree has a lower depth, while a deep tree has a higher depth. The maximum depth of the selection tree is a critical parameter that influences the version's complexity.

Impact of depth on Accuracy:

1. **Underfitting (Shallow Trees)**: when a decision tree is too shallow, it may not capture enough of the underlying patterns in the data. This can lead to high bias and low variance, resulting in poor accuracy on both the training and test sets. In such cases, increasing the tree depth can improve accuracy by allowing the tree to capture more complex patterns in the data.
2. **Overfitting (Deep Trees):** when a decision tree is too deep, it may memorize the training data instead of learning general patterns. This can lead to high variance and low bias, resulting in high accuracy on the training set but poor performance on the test set. In this case, reducing the tree depth can improve generalization and hence improve accuracy on unseen data.

The goal is to find the most accurate depth. When tuning the max depth parameter, it is important to keep in mind the size and complexity of the dataset.

Result and Discussion:

The custom decision tree on the Wine dataset achieved an accuracy of **96.3%**. This is a high accuracy, which indicates that the custom implementation is performing quite well on this particular dataset. The Sklearn Decision Tree achieved the same accuracy (**96.3%**) as the custom implementation. This is expected, as both implementations are using the same algorithm and a similar hyperparameter (max_depth=3), which means they should ideally produce similar results.

The custom decision tree on the Titanic dataset achieved a much lower accuracy (**80.97%**), compared to the Wine dataset. This result might be due to several reasons: data complexity, overfitting or underfitting (a depth=3 might not be sufficient for this dataset).