**Raport – 280462 – Bartosz Wacławiak**

**Dataset**

[Predict Student’s Dropout and Academic Success](https://www.kaggle.com/datasets/naveenkumar20bps1137/predict-students-dropout-and-academic-success)

**Project structure**

The project is structured into the following directories:

* data/ – contains the dataset in CSV format
* common/ - includes constants and helper functions used throughout the project
* notebooks/ – consists of Jupyter notebooks with all of tasks

**Dataset partioning**

In each notebook the dataset is divided into the following sets

* training set – 64%
* validation set - 16%
* test set - 20%

**Model training and prediction using a pipeline**

Following column transformer was used:

* for numerical features:
  + replacement of missing values with the median
  + standardization
* for categorical features:
  + replacement of missing values with the most frequent category
  + one-hot encoding

The following models from the scikit-learn libarary were used:

1. Logistic Regression (LogisticRegression)
2. Decision Tree Classifier (DecisionTreeClassifier)
3. Support Vector Machine (SVC)

Each model was trained and evaluated five times for accuracy on the training, validation, and test datasets.

1. Logistic Regression (LogisticRegression)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset | 1 | 2 | 3 | 4 | 5 |
| Train | 0.8100 | 0.8093 | 0.8100 | 0.8191 | 0.8093 |
| Validation | 0.7740 | 0.7797 | 0.7938 | 0.7429 | 0.7768 |
| Test | 0.7797 | 0.7864 | 0.7763 | 0.7853 | 0.7729 |

1. Decision Tree Classifier (DecisionTreeClassifier)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset | 1 | 2 | 3 | 4 | 5 |
| Train | 0.8001 | 0.8117 | 0.8025 | 0.8011 | 0.8061 |
| Validation | 0.7373 | 0.7218 | 0.7641 | 0.7302 | 0.7246 |
| Test | 0.7458 | 0.7435 | 0.7627 | 0.7277 | 0.7401 |

1. Support Vector Machine (SVC)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset | 1 | 2 | 3 | 4 | 5 |
| Train | 0.8365 | 0.8400 | 0.8410 | 0.8365 | 0.8301 |
| Validation | 0.7825 | 0.7458 | 0.7458 | 0.7571 | 0.7782 |
| Test | 0.7808 | 0.7842 | 0.7819 | 0.7785 | 0.7842 |

The Logistic Regression and SVC models achieve similar accuracies across training, validation, and test sets. On the other hand, the Decision Tree Classifier achieves worse results, which might be the matter of parametrization.

**Closed-form Linear Regression**

Linear regression has the following equation:

where

y – vector of observed values

X – matrix of observations (rows) and features (columns)

w – vector of weights

Then the normal equation to compute the optimal weights *w* directly:

**Linear Regression with Gradient Descent**

This implementation iteratively minimizes the MSE over the training data to optimize the weights.

The linear regression model assumes

The weights are iteratively updated using

Where the gradient for the const function is given by

**Linear Regression implementations**

Each linear regression implementation was trained and evaluated five times for mean squared error on the training, validation, and test datasets

1. Closed-form Linear Regession

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset | 1 | 2 | 3 | 4 | 5 |
| Train | 136.4140 | 135.5521 | 136.2144 | 137.0914 | 134.5934 |
| Validation | 128.7778 | 131.9047 | 130.4448 | 126.2236 | 135.9268 |
| Test | 131.2311 | 130.6790 | 130.6887 | 130.8080 | 130.3862 |

1. Linear Regression with Gradient Descent

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset | 1 | 2 | 3 | 4 | 5 |
| Train | 136.5241 | 135.7230 | 136.3940 | 137.2926 | 134.7437 |
| Validation | 128.9442 | 131.9764 | 130.1461 | 126.5491 | 136.1407 |
| Test | 131.4243 | 130.6763 | 130.1332 | 130.7191 | 130.5336 |

1. Sklearn Linear Regression

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset | 1 | 2 | 3 | 4 | 5 |
| Train | 136.4140 | 135.5521 | 136.2144 | 137.0914 | 134.5934 |
| Validation | 128.7778 | 131.9047 | 130.4448 | 126.2236 | 135.9268 |
| Test | 131.2311 | 130.6190 | 130.6887 | 130.8080 | 130.3862 |

All three implementations produce comparable results. However, the gradient descent approach turns out to be slightly less effective, which is expected due to its iterative nature and reliance on stochasticity. On the other hand, both the closed-form solution and scikit-learn's implementation yield identical results.

**Logistic Regression with Gradient Descent**

This implementation iteratively minimizes the binary cross-entropy loss over the training data to optimize the weights.

The logistic regression model assumes

where is the sigmoid function, X is the input matrix, and w is the weight vector.

The weights are iteratively updated using

Where the gradient for the const function is given by

Each logistic regression implementation was trained and evaluated five times for accuracy on the training, validation, and test datasets

1. Custom Logistic Regression with gradient descent

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset | 1 | 2 | 3 | 4 | 5 |
| Train | 0.8199 | 0.8124 | 0.8209 | 0.8244 | 0.8223 |
| Validation | 0.7910 | 0.8023 | 0.7881 | 0.7528 | 0.7726 |
| Test | 0.7571 | 0.7684 | 0.7661 | 0.7650 | 0.7684 |

1. Sklearn Logistic Regression

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset | 1 | 2 | 3 | 4 | 5 |
| Train | 0.7838 | 0.7838 | 0.7831 | 0.7817 | 0.7821 |
| Validation | 0.7655 | 0.7868 | 0.7655 | 0.7797 | 0.7952 |
| Test | 0.7537 | 0.7559 | 0.7514 | 0.7548 | 0.7571 |

The custom logic regression implementation with gradient descent provides slightly worse evaluations compared to the scikit-learn’s implementation, which might be the metter of parametrization.

**Linear Regression with PyTorch**

The model is built as a simple neural network with:

* a single linear layer (only one layer of neurons that perform linear transformation   
  y = Wx+ b)
* Mean Squared Error (MSE) as the loss function
* an optimizer such as Stochastic Gradient Descent (SGD)

The training loop consists of:

* forward pass through the model (returns output / prediction with the current weights)
* computation of the loss using a cost function (MSE)
* backward pass (backpropagation, gradient computation to determine how each parameter / weight contributed to the output / prediction)
* weight update via the optimizer (adjusts weights to reduce the loss)

Linear regression was trained and evaluated five times for mean squared error on the training, validation, and test datasets for GPU and CPU

1. CPU

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 |
| Training time | 3.98 | 3.21 | 3.90 | 3.89 | 3.88 |
| Train | 133.4656 | 136.5765 | 137.6624 | 138.1828 | 138.0577 |
| Validation | 151.2813 | 138.6728 | 135.2426 | 132.1579 | 132.6512 |
| Test | 121.7871 | 123.2026 | 124.0619 | 122.4109 | 122.2585 |

1. GPU

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 |
| Training time | 6.13 | 6.03 | 5.93 | 5.94 | 5.97 |
| Train | 133.5033 | 136.5796 | 137.6080 | 138.1756 | 138.0409 |
| Validation | 151.162277 | 138.6446 | 134.9460 | 132.2841 | 132.5421 |
| Test | 121.6101 | 123.3583 | 123.9044 | 122.5772 | 122.2799 |

The comparison between CPU and GPU execution reveals that both configurations achieve similar results in terms of MSE. The difference is negligible, confirming that the model behaves consistently regardless of the computational backend.

Interestingly, the GPU runtime is higher than the CPU runtime, which may seem counterintuitive at first. However, this can be explained by the overhead associated with data transfer between the CPU and GPU, which includes CUDA library setup or VRAM initialization. For small datasets and not complicated models like in this case, the GPU advantage is not fully utilized. Nevertheless, more complex computing problems are where GPU truly outperforms CPU.