⏱️ Classification benchmarks with Logistic Regression and Neural Networks

The project performs image classification using the CIFAR-10 dataset, downloaded and imported through the TensorFlow module. Image data is preprocessed: Specifically, images are greyscaled, pixel values are normalized, and 2D image array is flattened into 1D array. Numerical labels are converted to lexical, taken from Cifar10 docs. Classifier is trained and fit to the data. Predictions on test split are performed and classification report is (optionally) saved to the `out` directory. The project is comprised of two scripts, both utilizing scikit-learn for machine learning. The script `logistic_regression.py` utilizes the `LogisticRegression()` classifier, while the `neural_network.py` script utilizes the `MLPClassifier()`. Both scripts serve as pipelines for applying their respective model architectures to a classification task. In the case of the neural network, the loss curve is plotted and saved.

📈 Data

📋 Dataset

The project's image classification is based upon the CIFAR-10 dataset. The dataset contains 60.000 32x32 color images divided between 10 classes, it's divided into 50.000 training images and 10.000 test images (83/17). The dataset should be placed in the `in` directory, and can be sourced here.

📂 Project structure

```
└── image_classification_benchmarks
    ├── out
    │   ├── logistic_regression
    │   └── neural_network
    ├── src
    │   ├── logistic_regression.py
    │   ├── neural_network.py
    │   └── utilities.py
    │
    ├── setup_unix.sh
    ├── setup_win.sh
    ├── run_log.sh
    ├── run_neural.sh
    ├── requirements.txt
    └── README.md
```

⚙️ Setup

🐍 Dependencies

Please ensure you have the following dependencies installed on your system:

- **Python**: `version 3.12.3`

💾 Installation

1. Clone the repository

```
git clone https://github.com/apathriel/cds-vis-analytics
```

2. Navigate to the project directory

```
cd assignments
cd image_classification_benchmarks
```

3. Run the setup script to install dependencies, depending on OS.

```
bash setup_unix.sh
```

4. Activate virtual environment (OS-specific) and run main py scripts.

```
source env/bin/activate
python src/logistic_regression.py
```

🚀 Usage

After you have installed the necessary dependencies and set up your environment, you can run the scripts in this project. The main scripts contain pre-defined hyperparameters for the model responsible for the image classification, `LogisticRegression` and `MLPClassifier`. If run, the script will execute with the default parameters, identified through iterative testing and a summative grid search implemented through scikit-learn.

📊 Results

📝 Summary

The project's main scripts generate a each classification report using scikit-learn, which serves as an obvious point of comparison between the model architectures.

Please refer to the table below for a summative comparison of key evaluation metrics: (Weighted avg was selected, but effectively the same as macro avg, as classes have same split).

Model	Metric	Precision	Recall	F1-score
Logistic Regression	Weighted Avg	0.31	0.31	0.31
Neural Network	Weighted Avg	0.44	0.44	0.44

Please refer to the respective classification reports for the full reports.

📐 Hyperparameters

The results were produced using the following hyperparameters. They were identified and initiated through a mix of trial-and-error, summatively evaluated through a grid search, specifically implemented through scikit-learn's `GridSearchCV`.

Logistic Regression

Hyperparameter	Value
penalty	"l2"
solver	"saga"
C	0.001
max_iter	1000
tol	0.001
multi_class	"multinomial"

Neural Network

Hyperparameter	Value
hidden_layer_sizes	(256, 128)
activation	"relu"
solver	"adam"
alpha	0.05
max_iter	500
early_stopping	True
learning_rate_init	0.001

🔎 Interpretation

Looking at the F1-scores of 0.31 for the Logistic Regression model and 0.44 for the Neural Network: While `neural_network.py` takes longer to run, it performs substantially better on the image classification task. This embodies the shift from stastical models to neural networks. The neural network implemenation was quite simple, and the complexity, and likely performance, could be greatly increased. Implementing a deep learning CNN architecture with self-attention mechanisms would likely lead to substantial improvements.

📖 References

- Cifar10 dataset
- scikit-learn Logistic Regression

- scikit-learn MLPClassifier