Quantum Inspired Evolutionary Algorithms

1st Bulat Sharipov
b.sharipov@innopolis.university

2st Dinar Yakupov
d.yakupov@innopolis.university

3st Rail Sharipov
ra.sharipov@innopolis.university

I. Introduction

Training Machine Learning models involves optimizing both trainable and non-trainable parameters. The latter are commonly referred to as hyperparameters, while the former are simply called parameters. These parameters have a direct impact on the success of a model. Typically, trainable parameters are optimized using gradient descent, while hyperparameters are selected based on heuristics or manual tuning. Although gradient descent is efficient for optimizing parameters, it often converges to a local minimum, which can prevent the model from reaching its full potential. On the other hand, finding optimal hyperparameters requires exploring a potentially infinite set of combinations, making this a very challenging task.

To address both of these challenges, we explore the use of Quantum-Inspired Evolutionary Algorithms (QIEAs). These algorithms are motivated by principles from quantum mechanics and natural processes, and aim to optimize both trainable and non-trainable parameters effectively. In this project, we apply a QIEA to improve the training process of a Machine Learning model by overcoming limitations of traditional optimization techniques such as gradient descent and manual hyperparameter tuning.

II. Related Work

The core algorithm in our approach is the Quantum-Inspired Acromyrmex Evolutionary Algorithm, proposed by Montiel et al. [1]. This paper, recommended by Professor Muhammad Fahim for our project, introduces the reader to the basics of quantum operations and describes how these ideas are applied within an evolutionary algorithm. The algorithm can be viewed as a variant of the Genetic Algorithm, where solutions are represented and manipulated using concepts from quantum mechanics.

For example, candidate solutions are modeled as sets of qubits, which, when measured, collapse into specific binary solutions with certain probabilities. During the crossover phase, several quantum operations are used: the X-gate flips a bit's value (from 0 to 1 or vice versa), and the Hadamard gate places a bit in a state of superposition. These quantum operations enhance the algorithm's ability to explore the solution space more effectively.

The algorithm is named "Acromyrmex" after a species of ants, whose behavior inspired the structure of the population in the algorithm. Similar to an ant colony, the algorithm includes a queen, a selected group of males for reproduction, and a larger worker population, mimicking natural roles within the optimization process.

III. Methodology

To evaluate the effectiveness of quantum-inspired optimization, we implemented a Logistic Regression model entirely from scratch. Logistic Regression is a linear classifier that models the probability of a binary outcome using a sigmoid activation function applied to a linear combination of features. The model estimates weights $\mathbf{w}$ such that:

$$P(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{w}^\top \mathbf{x})}}$$

Training this model involves minimizing the negative log-likelihood loss function, optionally with regularization terms such as L1, L2, or elastic-net.

Optimization Methods. Our custom implementation supports two optimizers: (1) a standard gradient descent algorithm, and (2) a Quantum-Inspired Evolutionary Algorithm (QIEA). For gradient descent, we compute gradients of the loss with respect to the weights and iteratively update them using a fixed learning rate. Regularization is applied during gradient computation, where L1 adds a sparsity-inducing term, L2 penalizes large weights, and elastic-net combines both.

The evolutionary optimizer avoids gradients entirely. Instead, it treats the weights as a genome encoded as a binary string and applies quantum-inspired operators during a population-based search. Each individual in the population is a quantum chromosome composed of qubits, which are collapsed into binary values during measurement. Crossover is implemented via quantum gates, including the X-gate and Hadamard gate, allowing for superposition and non-classical recombination strategies.

Hyperparameter Tuning. We also developed a hyperparameter tuner using the same evolutionary framework. This tuner optimizes five key hyperparameters: optimizer type (gradient vs evolutionary), regularization type, regularization strength C, learning rate (or number of evolutionary runs), and maximum iteration count.

Hyperparameters are encoded as floating-point values, normalized and converted to a binary representation suitable for the QIEA. Each candidate configuration is evaluated by training a model on the training set and computing the AUC score on a validation set. The fitness function directly returns this AUC score, and the tuner selects the configuration yielding the highest validation performance.

This approach allows for a gradient-free search of the hyperparameter space, enabling the discovery of non-trivial configurations that may be missed by grid or random search

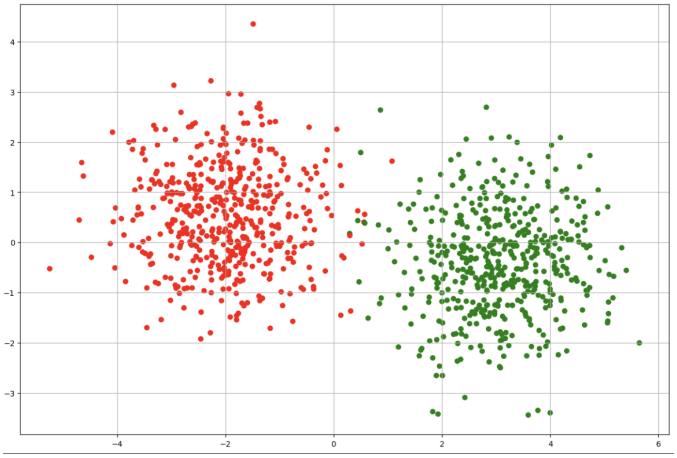

Fig. 1. Synthetic dataset generated with `make_blobs`

methods. However, due to the expensive nature of evolutionary optimization, especially when nested within model training, runtime remains a limitation for large-scale tuning.

IV. Github Link

The full source code, including the implementation of models, optimizers, and evaluation scripts, is available at the following repository: this link.

V. Experiments and Evaluation

The complete code for experiments and evaluations is located in the `evaluation` directory of the repository.

A. Synthetic Dataset

To initially test the functionality of our Logistic Regression implementation, we applied it to a synthetic dataset generated using `sklearn`'s `make_blobs` function. This dataset consists of two clusters of points in a two-dimensional space, each representing a different class. A visual representation of the dataset is shown in Fig. 1.

We tested both optimization methods on this dataset. The results are summarized below:

- Both optimizers converged quickly on the dataset.
- Both achieved an accuracy score of 0.99.
- However, the resulting decision boundaries were different. The evolutionary-based optimizer produced a decision boundary that was more perpendicular to the x-axis compared to the one generated by the gradient-based optimizer.

These differences in the decision boundary are illustrated in Fig. 2 and Fig. 3, which show the classifiers trained with gradient-based and evolutionary-based optimizers, respectively.

B. Titanic Dataset

As outlined in the project proposal, we also evaluated our models on a real-world dataset to observe their performance under practical conditions. For this purpose, we selected

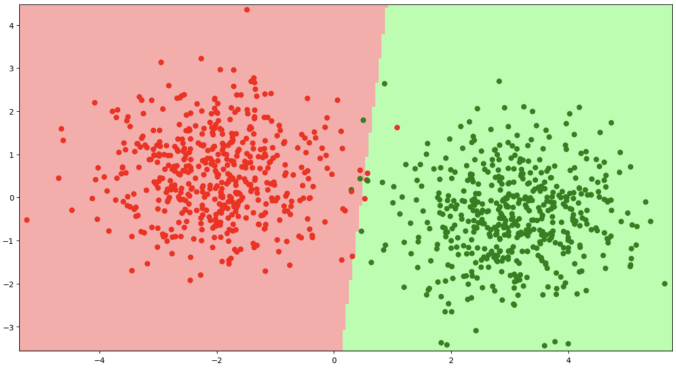

Fig. 2. Decision boundary from gradient-based optimizer

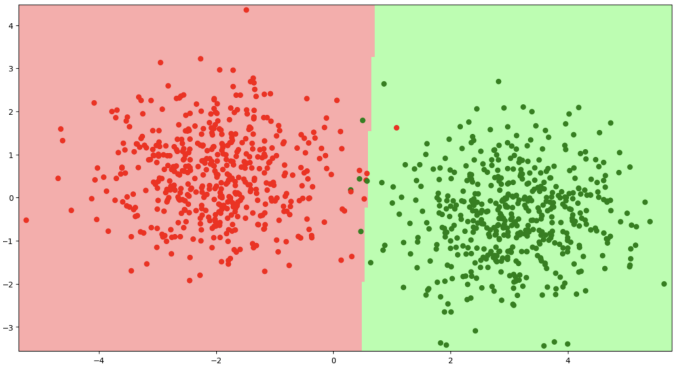

Fig. 3. Decision boundary from evolutionary-based optimizer

the well-known Kaggle Titanic dataset. This dataset contains passenger records, including attributes such as gender, class, ticket fare, and survival status. The goal is to predict whether a given passenger survived the Titanic disaster.

We applied both optimizers to this dataset. The observations were as follows:

- The gradient-based optimizer converged almost instantly.
- The evolutionary-based optimizer required more time to converge.
- Despite the longer runtime, the evolutionary optimizer achieved significantly better classification performance.

Specifically, the accuracy improved from 0.62 with the gradient-based optimizer to 0.82 using the evolutionary-based optimizer. Additionally, the Area Under the ROC Curve (AUC) also increased notably:

- Gradient-based optimizer: AUC score of 0.62.
- Evolutionary-based optimizer: AUC score of 0.87.

The ROC curves for both optimizers are displayed in Fig. 4 and Fig. 5. The classification reports generated using scikit-learn's `classification_report()` function are presented in Fig. 6 and Fig. 7.

VI. Analysis and Observations

As part of the project, we also implemented a hyperparameter tuner using the same Acromyrmex algorithm for the Logistic Regression model. The tuner was able to improve

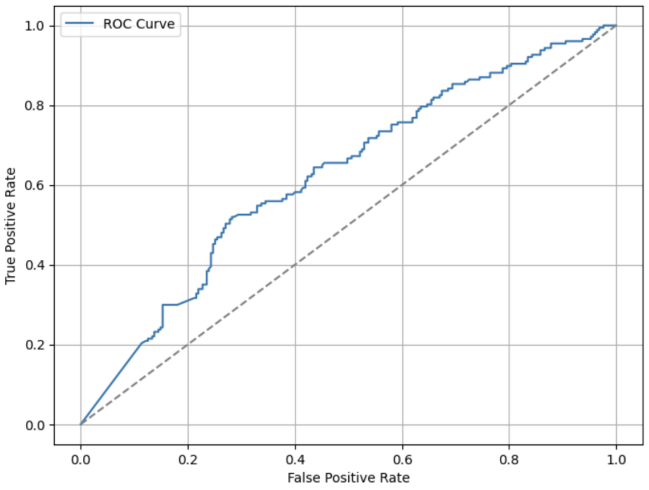

Fig. 4. ROC curve for gradient-based optimizer

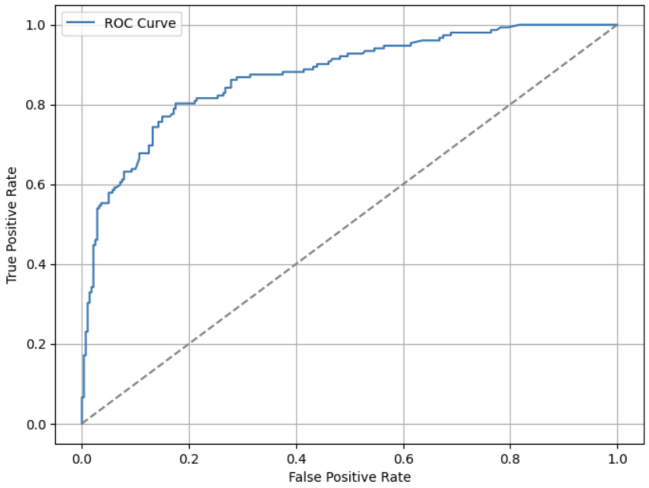

Fig. 5. ROC curve for evolutionary-based optimizer

	precision	recall	f1-score	support
0	0.68	0.65	0.67	255
1	0.53	0.56	0.54	177
accuracy			0.62	432
macro avg	0.61	0.61	0.61	432
weighted avg	0.62	0.62	0.62	432

Fig. 6. Classification report for gradient-based optimizer

	precision	recall	f1-score	support
0	0.86	0.87	0.86	280
1	0.75	0.73	0.74	152
accuracy			0.82	432
macro avg	0.80	0.80	0.80	432
weighted avg	0.82	0.82	0.82	432

Fig. 7. Classification report for evolutionary-based optimizer

model performance compared to the basic gradient-based Logistic Regression, achieving better classification scores by optimizing hyperparameters such as learning rate, regularization type, and iteration count. However, this improvement came at the cost of significantly increased runtime, especially when combined with the already slow evolutionary optimizer.

The following are our main observations:

- The evolutionary optimizer can be easily integrated into modern Machine Learning pipelines. Its implementation only requires adding a dedicated method for parameter updates within the model class.
- Evolutionary algorithms can be applied to a wide range of models through parameter encoding. In our setup, each parameter was represented using 16 bits, resulting in a binary string of length $16 \times 5 = 80$ bits for five parameters. This string was then decoded during optimization.
- The evolutionary optimizer consistently outperformed the gradient-based optimizer in terms of F1-score and AUC score.
- While the Acromyrmex algorithm generated more effective parameter sets, it required substantially more computational time to converge.
- The hyperparameter tuner based on evolutionary optimization was successful in improving performance over a standard Logistic Regression, but its runtime makes it less practical for large-scale or time-sensitive applications. Its combination with the evolutionary optimizer further amplifies this issue due to nested optimization loops.

VII. Conclusion

This project explored the application of a nature-inspired optimization method: the Quantum-Inspired Acromyrmex Evolutionary Algorithm. We successfully integrated the algorithm to train a Logistic Regression model, leading to noticeable improvements in classification performance compared to traditional gradient-based training.

Additionally, we developed a hyperparameter tuner using the same evolutionary framework. The tuner improved the model's performance by effectively navigating the hyperparameter space, particularly when paired with the gradient-based optimizer. However, the high runtime of the tuner, especially when used alongside the evolutionary optimizer, limits its practicality for more demanding use cases.

Moreover, the evolutionary optimizer introduces several of its own hyperparameters—such as population size, number

of elites, and crossover rate—which themselves may require tuning. This recursive complexity highlights both the flexibility and challenges of using such global search techniques.

Nonetheless, evolutionary tuning remains a promising direction, especially when used with fast training routines. Future work could explore integrating the tuner with lightweight models or accelerating the evolutionary process itself to make such approaches more scalable and applicable in real-world scenarios.

REFERENCES

[1] F. Montiel et al., "Quantum-inspired Acromyrmex Evolutionary Algorithm for Global Optimization," *Scientific Reports*, vol. 9, Article 48409, 2019, doi:10.1038/s41598-019-48409-5.