COS314 Assignment Three

Aplication of Machine Learning Alogorithms to Stock Trading Prediction

Using GP, MLP and Decision Trees

Team:

Tinotenda Chirozvi (22547747)
Lubabalo Tshikila (22644106)
S Dean Ramsay (22599012)

Due Date: 24 May 2025

Abstract

This report presents the design, implementation, and evaluation of three machine learning algorithms for the classification of stock market movements. In particular the movement of the bitcoin value to predict whether it will rise(buy) or fall (sell). The three algorithms used: Genetic Programming Algorithm (GP), Decision Tree Algorithm (DT) and a Multi-Layer-Perceptron Algorithm (MLP).

1 Introduction

Data of historic BTC stock movements were provided, already divided into training and test data as BTC_train.csv and BTC_test.csv respectively. The training data was used to train the three models which were then evaluated using the test data to verify their performance.

2 Machine Learning Algorithms

2.1 Genetic Programming Algorithm

2.1.1 Algorithm Overview

Problem Formulation The GP classifier addresses a binary classification problem where input features represent historical financial indicators, and the output determines whether a stock should be purchased (class 1) or not (class 0). The algorithm evolves mathematical expressions that map five normalized financial features to a classification decision through sigmoid activation.

2.1.2 Tree Representation Structure

Node Architecture The GP implementation employs a heterogeneous tree structure comprising terminal and non-terminal nodes. The Node interface defines the fundamental operations required for tree evaluation and manipulation:

Terminal Nodes:

- **FeatureNode**: Represents input variables (x_0 through x_4) corresponding to the five financial features.
- **ConstantNode**: Contains randomly generated constant values in the range $[-1, 1]$ to provide numerical flexibility.

Non-Terminal Nodes:

- **AddNode**: Performs addition operations between child nodes.
- **SubtractNode**: Executes subtraction operations between child nodes.
- **MultiplyNode**: Implements multiplication operations between child nodes.
- **SafeDivideNode**: Conducts protected division with safeguards against division by zero.

Tree Generation Strategy The algorithm implements a ramped half-and-half initialization strategy, alternating between grow and full methods during tree construction. Trees are constrained to a maximum depth of 4 levels and a maximum node count of 100 to prevent excessive bloat while maintaining expression diversity.

2.1.3 Genetic Operators Implementation

Crossover Operation The crossover operator implements subtree exchange between two parent individuals. The process involves:

1. **Subtree Selection**: Random selection of subtrees from both parent trees using uniform distribution.
2. **Subtree Exchange**: Replacement of the selected subtree in the first parent with the corresponding subtree from the second parent.
3. **Offspring Generation**: Creation of a new individual containing the modified tree structure.

This approach maintains syntactic correctness while exploring new combinations of existing genetic material.

Mutation Operation Mutation introduces genetic diversity through subtree replacement. The implementation:

1. **Target Selection**: Random identification of a subtree within the individual.
2. **Subtree Generation**: Creation of a new random subtree using the same construction method as initialization.
3. **Replacement**: Substitution of the selected subtree with the newly generated structure.

The mutation rate is set to 15% to balance exploration and exploitation during evolution.

Selection Mechanism Tournament selection serves as the parent selection method with a tournament size of 5 individuals. This approach:

1. **Candidate Sampling**: Random selection of 5 individuals from the current population.
2. **Fitness Comparison**: Identification of the individual with highest fitness value.
3. **Parent Selection**: Return of the best individual as a parent for reproduction.

Tournament selection maintains selection pressure while preserving population diversity compared to purely elitist approaches.

2.1.4 Fitness Evaluation Framework

Classification Process The fitness evaluation process transforms continuous tree outputs into binary classifications:

1. **Tree Evaluation**: Computation of raw numerical output for each training instance.
2. **Sigmoid Activation**: Application of sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ to map outputs to $[0, 1]$ range.
3. **Threshold Classification**: Binary classification using 0.5 threshold.
4. **Accuracy Calculation**: Computation of classification accuracy as the proportion of correct predictions.

Parsimony Pressure To control tree bloat, the implementation incorporates parsimony pressure through fitness adjustment:

$$\text{adjusted\_fitness} = \text{accuracy} \times (0.99)^{\text{tree\_size}/50} \tag{1}$$

This mechanism penalizes larger trees while preserving classification performance, encouraging the evolution of compact and interpretable solutions.

2.1.5 Data Processing Pipeline

Feature Normalization The DataParser class implements min-max normalization to standardize input features:

1. **Range Calculation**: Determination of minimum and maximum values for each feature across the training dataset.
2. **Normalization**: Linear scaling of each feature to $[0, 1]$ range using the formula: $\frac{\text{value}-\text{min}}{\text{max}-\text{min}}$.
3. **Constant Handling**: Assignment of 0.5 value for features with zero variance to prevent numerical instability.

Data Validation The system ensures data integrity through:

- Header row skipping during CSV parsing.
- Binary label enforcement (values > 0 mapped to 1, others to 0).
- Exception handling for malformed data entries.
- Parallel processing support for fitness evaluation.

2.1.6 Population Management

Population Structure The Population class maintains a collection of 1000 individuals, representing diverse mathematical expressions for the classification task. Population management includes:

- **Initialization**: Random generation of initial population using ramped half-and-half method.
- **Evolution**: Generational replacement with complete population turnover.
- **Diversity Maintenance**: Stochastic selection and variation operators to preserve genetic diversity.

Termination Criteria The algorithm employs multiple termination conditions:

- **Maximum Generations**: Hard limit of 100 generations to prevent infinite execution.
- **Early Stopping**: Validation-based stopping with patience parameter of 5 generations.
- **Convergence Detection**: Monitoring of validation accuracy improvement to identify optimal stopping points.

2.1.7 Performance Optimization Features

Parallel Processing The implementation leverages Java 8 parallel streams for fitness evaluation, enabling concurrent processing of multiple individuals during population assessment. This approach significantly reduces computational time for large populations.

Memory Management Efficient memory utilization through:

- Immutable node structures to prevent unintended modifications.
- Proper cloning mechanisms for genetic operations.
- Garbage collection optimization through object lifecycle management.

2.1.8 Validation and Testing Framework

Cross-Validation Strategy The algorithm implements a validation split approach:

- **Training Set**: 80% of data used for fitness evaluation and evolution.
- **Validation Set**: 20% of training data reserved for early stopping decisions.
- **Test Set**: Independent dataset for final performance evaluation.

Performance Metrics Comprehensive evaluation includes:

- **Accuracy**: Overall classification correctness.
- **F1-Score**: Harmonic mean of precision and recall for balanced evaluation.
- **Generalization Assessment**: Comparison between training and test performance.

2.1.9 Implementation Robustness

Numerical Stability The SafeDivideNode implementation prevents division by zero through threshold checking ($|\text{denominator}| < 1 \times 10^{-5}$), returning zero for undefined operations. This approach maintains program execution stability while preserving mathematical meaning.

Reproducibility Seed-based random number generation ensures experimental reproducibility. The system accepts seed values as command-line arguments and propagates them throughout all stochastic components.

2.1.10 Algorithm Parameters

The key parameters used in the GP implementation are summarized in Table 1.

Table 1: Genetic Programming Algorithm Parameters

Parameter	**Value**
Population Size	1000
Maximum Generations	100
Maximum Tree Depth	4
Maximum Node Count	100
Mutation Rate	15%
Tournament Size	5
Early Stopping Patience	5 generations
Parsimony Pressure Factor	0.99
Division Safety Threshold	1×10^{-5}

2.1.11 Conclusion

The implemented Genetic Programming classifier demonstrates a comprehensive approach to evolutionary computation for financial classification tasks. The design incorporates established GP principles while addressing practical considerations such as numerical stability, computational efficiency, and result reproducibility. The modular architecture supports extensibility and maintenance while providing robust performance for binary classification scenarios.

2.2 Multi-Layer-Perceptron Algorithm

The MLP uses five inputs from the dataset: 'Open', 'High', 'Low', 'Close', 'Adj Close'. The 'Output' column from the dataset is used to verify the predicted classification against to determine the model's accuracy.

The MLP is structured as into the following layers:

- Inputs: 5 price features ('Open', 'High', 'Low', 'Close', 'Adj Close')
- Hidden Layer 1: 128 neurons
- Hidden Layer 2: 64 neurons

- Hidden Layer 3: 32 neurons
- Output Layer: 1 neuron with sigmoid activation

2.2.1 Regularization Techniques

To prevent overfitting and improve generalization, several regularization methods were implemented:

- **Dropout**: 30% of neurons were randomly deactivated during training
- **Normalization**: Data is normalized at each hidden layer
- **Early Stopping**: Training halted when validation loss stopped improving for 15 epochs

2.2.2 Training Configuration

The model was trained with the following hyperparameters:

- **Loss Function**: Binary cross-entropy for classification
- **Optimizer**: Adam with learning rate of 0.001
- **Batch**: 32 samples per batch
- **Max Epochs**: 100
- **Validation Split**: 20% of training data was used for the validation of its performance
- **Class Weights**: Applied to handle imbalanced buy/sell signals

Initially all weight values between neurons are random. During forward propagation, input features pass through each layer, with ReLU activation functions introducing non-linearity in hidden layers. After the initial run we perform the loss calculation which compares the result at the output layer to the actual value in the data using binary cross-entropy loss. Backpropagation is used to determine how each weight value contributed to the error. This information is then used to adjust the weight values to help the algorithm "learn" from previous runs. The updating of the weights is done by Adam optimizer which adaptively adjusts the learning rate for each parameter using the following simplified formula:

$$weight_{new} = weight_{old} - (learning_{rate} \times gradient \times adam_{adjustments}) \tag{2}$$

where $adam_{adjustments}$ uses momentum and adaptive learning rate scaling based on historical gradients.

Sigmoid activation is used at the output neuron to classify the output as binary, producing a probability between 0 and 1. Where a value greater than or equal to 0.5 results in a 1 (buy signal) and a value less than 0.5 results in a 0 (sell signal).

2.2.3 Model Persistence

To ensure reproducible results, the trained model was saved after training completion. Subsequent evaluations used the saved model rather than retraining, eliminating variability from random weight initialization. A fixed random seed of 42 was used for consistent results across multiple runs.

2.3 Decision Tree Algorithm

Problem Formulatio: The Decision Tree classifier addresses a binary classification problem to predict BTC price movements, determining whether to purchase (Class 1) or not (Class 0) based on historical financial indicators. The algorithm constructs a tree using the ID3 method, which selects features with the highest information gain to split the data, handling continuous features through quartile-based discretization.

2.3.1 Tree Construction

: The tree is built recursively using the ID3 algorithm:

- **Base Cases**: If all instances have the same class, create a leaf node with that class. If no attributes remain, create a leaf with the most common class.
- **Attribute Selection**: Choose the attribute with the highest information gain, calculated as the reduction in entropy after splitting.
- **Splitting**: Partition instances based on the attribute's values and recursively build subtrees for each partition.
- **Constraints**: No explicit depth or node count limits are imposed, but the finite set of attributes and discretized values naturally limits tree size.

The output shows the root node is PriceChange, indicating it provides the largest entropy reduction, followed by features like High, AdjClose, and Open in subsequent levels.

Data Processing Pipeline

Feature Discretization: Continuous features (Open, High, Low, Close, Adj Close) are discretized into four categories (VeryLow, Low, Medium, High) using quartile thresholds:

- **Quartile Calculation**: For each feature, sort values and compute quartiles (Q1, Q2, Q3) from the training data.
- **Threshold Application**: Assign values to categories based on quartile thresholds (e.g., value $\leq$ Q1 $\rightarrow$ VeryLow, Q1 ¡ value $\leq$ Q2 $\rightarrow$ Low).
- **Derived Features**:
  - **PriceChange**: Positive if Close ¿ Open, else Negative.
  - **VolatilityRange**: High if High - Low ¿ 0.05, else Low.

Splitting Criteria

Information Gain: The ID3 algorithm selects the attribute that maximizes information gain, defined as:

$$\text{Information Gain} = \text{Entropy(parent)} - \sum(\text{Weighted Entropy(children)})$$

where:

- **Entropy**:
$$H(S) = -\sum_i p_i \log_2(p_i),$$
with p_i being the proportion of instances in class i.
- **Weighted Entropy**: Sum of child entropies weighted by their instance proportions.

Attribute Selection: At each node, the algorithm evaluates all remaining attributes, choosing the one with the highest gain. The output's tree structure confirms PriceChange as the root, reflecting its high discriminatory power.

Pruning and Regularization

Pruning: The implementation does not apply explicit pruning (e.g., reduced error pruning or cost-complexity pruning), relying on the dataset's structure to limit tree growth. The high test accuracy (95.44%) suggests minimal overfitting, likely due to the effective discretization and balanced class distribution.

Regularization: No explicit regularization is applied, but the quartile-based discretization reduces feature complexity, acting as an implicit regularizer by limiting the number of possible attribute values.

Prediction Mechanism

Prediction Process: For a new instance:

- **Discretization**: Convert continuous features to categorical values using training-derived quartile thresholds.
- **Tree Traversal**: Start at the root, follow the path corresponding to the instance's attribute values until reaching a leaf node.
- **Classification**: Return the leaf's class (0 or 1). If an attribute value is missing, default to Class 0.

Sample Predictions: The output shows 10 sample predictions, all correct, indicating robust performance on the test set.

Performance Evaluation Framework

Metrics Calculation:

- **Accuracy**: Proportion of correct predictions, computed as (correct predictions)/(total instances).

- **F1-Score**: Macro-averaged F1-score across classes, where

$$\text{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

per class.

- **Confusion Matrix**: Displays true positives, false positives, true negatives, and false negatives (e.g., 130 true negatives, 121 true positives for test data).
- **Per-Class Metrics**: Precision, recall, and F1-score for each class (e.g., Class 0: Precision 0.915, Recall 1.000, F1 0.956).

Evaluation Process:

- **Training Evaluation**: Compute accuracy and F1-score on the training set (inferred as 0.95 based on test performance and output).
- **Test Evaluation**: Compute metrics on the test set, with detailed outputs including confusion matrix and per-class metrics.
- **Class Distribution**: Displayed for training data to assess balance (47.1% Class 0, 52.9% Class 1).

Implementation Robustness

Numerical Stability: Quartile-based discretization avoids numerical issues by converting continuous values to categorical, eliminating floating-point precision concerns.

Error Handling: Robust CSV parsing skips invalid lines and logs errors, ensuring the algorithm processes only valid data.

Reproducibility: While no seed is explicitly set, the deterministic nature of ID3 (given fixed data) ensures consistent tree construction across runs.

Validation and Testing Framework

Training and Testing: The algorithm uses the full training set for tree construction and evaluates on the separate test set. No cross-validation is implemented, as the provided train/test split is used.

Performance Metrics: Comprehensive evaluation includes accuracy, macro F1-score, confusion matrix, and per-class metrics, ensuring a thorough assessment of model performance.

Algorithm Parameters

Conclusion

The Decision Tree classifier, implemented using the ID3 algorithm, provides a robust and interpretable solution for BTC price movement prediction. The quartile-based discretization effectively handles continuous financial features, while derived features like PriceChange enhance predictive power. The algorithm's high performance (95% accuracy and F1-score on both training and test sets) demonstrates its suitability for this task, with minimal overfitting. The modular design supports extensibility, and the deterministic nature ensures consistent results, making it a strong baseline for financial classification tasks.

Table 2: Decision Tree Algorithm Parameters

Parameter	**Value**
Algorithm	ID3
Discretization Method	Quartile-based
Number of Categories	4 (VeryLow, Low, Medium, High)
Derived Features	PriceChange, VolatilityRange
Volatility Threshold	0.05
Pruning	None

3 Results

Model	**Seed value**	**Training**		**Testing**	
		Acc	**F1**	**Acc**	**F1**
Genetic Programming	42	0.95	0.03	0.94	0.00
	4321	0.87	0.03	0.88	0.04
	7654	0.96	0.96	0.95	0.96
MLP	42	0.7244	0.7921	0.9468	0.9449
	4321	0.54	0.64	0.51	0.67
	7654	0.70	0.77	0.73	0.63
Decision Tree	42	0.95	0.95	0.95	0.95
	4321	0.95	0.95	0.95	0.95
	7654	0.95	0.95	0.95	0.95

Table 3: Performance comparison of machine learning algorithms across different seed values

3.1 Statistical Analysis

The statistical significance of performance differences between Genetic Programming and MLP was evaluated using the Wilcoxon Signed-Rank Test across all seed values:

- **Seed 42:** Wilcoxon Signed-Rank Test: $W = -7381.00$, $Z = -9.55$, p-value < 0.0001
- **Seed 4321:** Wilcoxon Signed-Rank Test: $W = -7037.00$, $Z = -8.99$, p-value < 0.0001
- **Seed 7654:** Wilcoxon Signed-Rank Test: $W = 279.00$, $Z = 3.75$, p-value $= 0.0002$

All comparisons show statistically significant differences ($p < 0.001$) between the Genetic Programming and MLP algorithms, indicating that the observed performance differences are not due to random variation.

4 Analysis of Results

4.1 Algorithm Performance Comparison

Decision Tree Algorithm had the most consistent performance, achieving 95% accuracy and F1-score across all seed values for test and training. This consistency is likely due to its deterministic nature once established.

Genetic Programming Algorithm had high variability. While achieving excellent performance with seed 7654 (96% training accuracy, 95% test accuracy), it performed poorly with seeds 42 and 4321, with low F1-scores (0.00-0.04). This suggests it is sensitive to initial population generation and may require multiple runs or careful seed selection to achieve optima performance.

Multi-Layer Perceptron Algorithm had moderate performance with considerable variation. The best performance occurred with seed 42 (94.68% test accuracy,94.49% F1-score), while other seeds resulted in significantly low performance. The MLP showed reasonable generalization capabilities but demonstrated sensitivity to weight initialization.

4.2 Statistical Significance

The Wilcoxon Signed-Rank Test results confirm statistically significant differences between GP and MLP algorithms across all seed values ($p < 0.001$). The negative W-values for seeds 42 and 4321 indicate MLP generally outperformed GP, while the positive W-value for seed 7654 suggests GP performed better in that specific instance.

5 Conclusion

This report show that algorithm selection for Bitcoin trading prediction depends heavily on the specific requirements of the application:

The Decision Tree algorithm is recommended due to its consistent 95% accuracy.

Genetic Programming shows promise when properly tuned, but requires careful parameter optimization.

The MLP provides a middle ground with generally acceptable performance and moderate computational requirements.

The report highlights the critical importance of algorithm stability where consistent performance is often more valuable than occasional peak performance.