Project Report: Expert System for Restaurant Recommendation

1. Introduction:

The objective of this project is to design and implement an expert system for restaurant recommendation, leveraging artificial intelligence techniques to assist users in making informed choices when selecting restaurants. The expert system aims to emulate the decision-making capabilities of a human restaurant expert and provide personalised recommendations based on user preferences, location, cuisine budget, and other relevant factors. This particular implementation also has been incorporated particularly for the BITS Hyderabad restaurants and food outlets, and the knowledge base has been upgraded as such.

2. Problem Statement:

The expert system will be developed to address the challenge of helping users(students) find suitable restaurants based on their individual preferences, dietary requirements, budget constraints, and location. The system will take into account various criteria to provide recommendations that align with the user's taste and needs.

3. System Architecture:

The architecture of the restaurant recommendation expert system consists of the following components:

- Knowledge Base: Contains a database of college restaurants, including their attributes such as cuisine type, price range, ratings, and payment methods.
- Inference Engine: The core component responsible for reasoning and decision-making. It utilises the knowledge base and user inputs to generate restaurant recommendations.
- User Interface: A user-friendly interface allowing users to input their preferences, location, and other relevant details to receive personalised restaurant suggestions.

4. Methodology:

The development process for the expert system involves the following key steps:

- Data Collection: Gather restaurant data from various sources, including online restaurant directories, user reviews, and official websites.
- Knowledge Representation: Structure the collected data into a suitable format for the knowledge base, incorporating attributes such as cuisine type, location, price range, and user ratings.
- Rule-Based Reasoning: Develop a set of rules that guide the inference engine to match user preferences with restaurants in the knowledge base effectively.
- User Profiling: Implement a mechanism to learn and update user preferences over time, refining the recommendation process with each interaction.
- Testing and Validation: Conduct extensive testing using sample user inputs and evaluate the accuracy and relevance of the generated restaurant recommendations.
- Deployment: Deploy the expert system as a user-facing application for real-world usage.

5. Implementation Details:

The restaurant recommendation expert system will be implemented using python,, and the rule-based reasoning approach will be used to generate recommendations. The knowledge base will be stored in a structured format, possibly utilising databases or ontology-based representations.

6. Results and Discussion:

The restaurant recommendation system achieved its initial targets, particularly as creating a central campus wide restaurant and food outlet recommendation software which is accessible and can be used by any student, faculty or otherwise general public if they wish to do so. The expert system still doesn't completely reflect what is sold at every single restaurant and whether a particular food item would be available at a certain outlet or not. That said, it is the only campus restaurant and food outlet recommendation software system available to the public

and does provide a lot of functionality to students or faculties who are not well versed with menus and food outlet specifications.

7. Future Work:

Incorporating natural language processing (NLP) to support more conversational interactions with the system, integrating user reviews and social media data to improve recommendations, and expanding the system to cover additional geographical locations.

Academic Report on Genetic Algorithm for Feature Selection in Stock Price Prediction

Introduction:
This academic report discusses a Python implementation of a Genetic Algorithm (GA) applied to feature selection for stock price prediction. The algorithm uses historical stock data of Apple Inc. (AAPL) from the 'AAPL.csv' file to train and evolve a population of feature subsets using GA. The goal is to select the most relevant features that optimize the prediction performance of stock prices. The code uses popular Python libraries such as Pandas, NumPy, Matplotlib, and scikit-learn.

Genetic Algorithm for Feature Selection:
Genetic Algorithms are a class of optimization algorithms inspired by natural selection and genetics. They are commonly used for feature selection in machine learning tasks to find the most relevant subset of features from a high-dimensional dataset. The GA implemented in this code aims to find an optimal feature subset that maximizes the correlation coefficient between the predicted stock prices and the actual stock prices.

Implementation Details:
1. The code begins by importing necessary libraries and loading the historical AAPL stock data from the 'AAPL.csv' file. Irrelevant columns such as 'Date', 'Adj Close', and 'Volume' are dropped from the dataset.
2. The GeneticAlgorithm class is defined, which takes parameters like population size, number of iterations, and mutation rate for the GA.
3. The `fit` method of the GeneticAlgorithm class is the main loop where the GA runs for a specified number of iterations.
4. In each iteration, a population of feature subsets is randomly initialized with floating-point values between 0 and 1.

5. The fitness of each individual in the population is evaluated using the correlation coefficient between the predicted stock prices and the actual stock prices.
6. Selection, crossover, and mutation operations are performed to evolve the population to the next generation.
7. After completing the specified number of iterations, the GA returns the best feature subset found during the evolution process.

Results and Evaluation:
The main loop of the GA records the fitness and predicted stock prices for each iteration. These values are stored in lists `fitness_return` and `y_pred_return`, respectively.

Two plots are generated to visualize the convergence of the algorithm:
1. Plot of Mean Absolute Error (MAE) between predicted stock prices and actual stock prices across iterations.
2. Plot of mean fitness (correlation coefficient) of the best individuals across iterations.

Discussion:
The first plot demonstrates how the GA evolves over iterations, reducing the MAE and improving the stock price prediction. As the GA iteratively improves the feature subset, the MAE between the predicted and actual stock prices decreases.

The second plot illustrates how the correlation coefficient, which is the fitness function of the GA, improves over iterations. As expected, the GA converges towards feature subsets that better correlate with the actual stock prices.

Prediction:
To demonstrate the application of the best feature subset found by the GA, a sample input `[14.055000, 14.830357, 14.017857]` is used to predict the stock price. The dot product of this input with the best feature subset yields the predicted stock price, which is approximately `21.92`.

Conclusion:
The implemented Genetic Algorithm successfully performs feature selection for stock price prediction using historical Apple Inc. stock data. By evolving a population of feature subsets over iterations, the GA identifies the most relevant features that optimize the prediction performance. The convergence plots demonstrate the improvement in prediction accuracy as the GA progresses. The best feature subset is then used to predict the stock price for a given input, showcasing the practical utility of the algorithm.

Academic Report on Linear Regression with Gradient Descent for Stock Price Prediction

Introduction:
This academic report presents an implementation of linear regression using the gradient descent algorithm to predict stock prices. The dataset used in this analysis contains historical stock data of Apple Inc. (AAPL) from the 'AAPL.csv' file. The goal is to predict the closing stock prices based on the 'Open', 'High', and 'Low' prices. The code employs Python and popular data science libraries like Pandas, NumPy, and Matplotlib.

Data Preprocessing:
The code begins by loading the AAPL stock dataset into a Pandas DataFrame and removing irrelevant columns ('Date', 'Adj Close', 'Volume') to focus on the features of interest.

Feature Normalization:
Before applying the gradient descent algorithm, feature normalization is performed to ensure that all features have similar scales. The `feature_normalize` function is defined to calculate the mean and standard deviation of each feature and then normalize the data accordingly. This step is crucial for gradient descent to converge effectively and avoid potential issues with the optimization process.

Cost Function:
The `compute_cost` function calculates the mean squared error, which serves as the cost function for linear regression. The cost function measures the discrepancy between the predicted 'Close' stock prices and the actual 'Close' stock prices based on the current model parameters (theta).

Gradient Descent:
The gradient descent algorithm is implemented in the `gradient_descent` function. It iteratively updates the model parameters (theta) to minimize the cost function. The algorithm calculates the gradient of the cost function with respect to each parameter and uses the learning rate (alpha) to control the step size of the updates. By performing multiple iterations, the algorithm converges towards the optimal parameters that best fit the data.

Selecting Learning Rates:
To ensure the effectiveness of the gradient descent, the code loops over different learning rates (0.3, 0.1, 0.03, and 0.01). For each learning rate, it runs the gradient descent algorithm for 200 iterations and plots the convergence of the cost function. This visual representation helps in selecting an appropriate learning rate that balances convergence speed and stability.

Training with Final Learning Rate:
After selecting the best learning rate (alpha = 0.01), the code performs the final training using this learning rate. It runs the gradient descent algorithm for 1500 iterations to optimize the model parameters (theta). The result is an accurate model capable of predicting 'Close' stock prices based on the 'Open', 'High', and 'Low' prices.

Making Predictions:
To demonstrate the practical application of the trained model, the code takes a sample input consisting of 'Open', 'High', and 'Low' stock prices. The input is normalized using the previously calculated mean and standard deviation. The trained model's optimized parameters (theta) are then utilized to predict the 'Close' stock price for the given input.

Conclusion:
This report presents a successful implementation of linear regression with the gradient descent algorithm for stock price prediction. The code efficiently learns the relationships between the 'Open', 'High', and 'Low' prices and the 'Close' stock prices of Apple Inc. based on historical data. The visualization of the cost function convergence aids in choosing an appropriate learning rate, ensuring a stable and efficient optimization process.

However, it is essential to consider that linear regression is a basic model, and stock price prediction is a complex task influenced by numerous factors. As such, more sophisticated models and evaluation metrics are required for accurate predictions in real-world financial scenarios.

Academic Report on Particle Swarm Optimization (PSO) for Stock Price Prediction

Introduction:
This academic report explores the application of Particle Swarm Optimization (PSO) to predict stock prices using historical data of Apple Inc. (AAPL). PSO is a metaheuristic optimization algorithm inspired by the collective behavior of social organisms, such as flocks of birds or schools of fish. The algorithm aims to optimize a fitness function by iteratively searching for the best set of model parameters to minimize the prediction error. The Python code utilizes Pandas, NumPy, and Matplotlib libraries to implement the PSO algorithm.

Data Preprocessing:
The code begins by loading the 'AAPL.csv' dataset into a Pandas DataFrame and removing irrelevant columns ('Date', 'Adj Close', 'Volume', 'Close'). The remaining columns, representing the features, are extracted as 'X', and the target variable 'Close' is assigned to 'y'.

Particle Swarm Optimization (PSO) Implementation:
The core PSO algorithm is implemented in the `pso` function, which takes the number of particles, the number of iterations, cognitive and social parameters (c1 and c2), inertia weight (w), features 'X', and target 'y' as input. PSO initializes a population of particles with random positions and velocities and searches for optimal solutions through iterative updates.

Particle Initialization and Movement:
The function initializes particles' positions randomly and velocities with zeros. It then updates particle velocities based on cognitive and social components, along with the inertia weight. The updated velocities guide the particles towards potentially better solutions in the solution space.

Fitness Evaluation:
For each particle, the function calculates the fitness, which is the mean squared error between the actual 'Close' stock prices and the predicted stock prices based on the particle's current position (model parameters). The fitness evaluation guides the particles to regions with lower prediction errors.

Updating Best Positions and Global Best:
PSO keeps track of the best positions and corresponding scores (fitness) for each particle and globally. These best positions represent the individual and global best solutions found during the optimization process.

PSO Iterations:
The algorithm iterates through multiple generations, each time updating the particles' positions and evaluating their fitness. The global best position and score are continually updated based on the best solutions found so far.

Convergence Visualization:
To assess the convergence behavior of the PSO algorithm, the code plots the error (mean squared error) over iterations. The plot helps in understanding how the prediction error evolves with each iteration and whether the algorithm converges to a stable solution.

Selecting Number of Particles:
To investigate the effect of the number of particles on the algorithm's performance, the code loops through different numbers of particles (25, 50, 100, and 200). For each case, it records the error convergence over iterations and plots the results. This analysis allows selecting an appropriate number of particles that balance computational efficiency and optimization effectiveness.

Conclusion:
This academic report demonstrates the application of Particle Swarm Optimization (PSO) for stock price prediction using historical Apple Inc. stock data. The PSO algorithm efficiently searches for the optimal model parameters that minimize the prediction error. The convergence visualization and analysis of different numbers of particles provide insights into the algorithm's performance and help in selecting suitable hyperparameters.

It is essential to consider that PSO is a stochastic optimization algorithm and may converge to different solutions for different runs. Therefore, conducting multiple runs and considering the average performance would further enhance the robustness of the optimization.

Future research may involve exploring alternative optimization algorithms, such as Genetic Algorithms or Differential Evolution, and assessing their performance in stock price prediction tasks. Additionally, incorporating additional relevant features or exploring more sophisticated machine learning models could lead to further improvements in prediction accuracy.

Please note that while PSO is a powerful optimization algorithm, predicting stock prices remains a challenging task due to the complex and dynamic nature of financial markets. Caution should be exercised while interpreting the results, and rigorous evaluation on unseen data is essential before applying the model to real-world financial decision-making.

Academic Report on Artificial Bee Colony Algorithm for Stock Price Prediction

Introduction:
This academic report delves into the application of the Artificial Bee Colony (ABC) algorithm for stock price prediction using historical data of Apple Inc. (AAPL) obtained from the 'AAPL.csv' file. ABC is an optimization algorithm inspired by the foraging behavior of honey bees. It aims to find the optimal set of model parameters for linear regression, minimizing the mean squared error (MSE) between the actual 'Close' stock prices and the predicted prices. The Python code employs Pandas, NumPy, and Matplotlib libraries to implement the ABC algorithm.

Data Preprocessing:
The code loads the 'AAPL.csv' dataset into a Pandas DataFrame and removes irrelevant columns ('Date', 'Adj Close', 'Volume', 'Close'). The remaining columns, representing the features, are extracted as 'X', and the target variable 'Close' is assigned to 'y'.

Fitness Function:
The `fitness_function` calculates the mean squared error (MSE) between the actual 'Close' stock prices and the predicted prices based on the current set of model parameters (weights 'w' and bias 'b'). The MSE serves as the fitness function for the ABC algorithm, guiding the search for optimal solutions.

Artificial Bee Colony (ABC) Implementation:
The core ABC algorithm is implemented in the `abc` function, which takes 'X', 'y', population size, maximum number of iterations (limit), lower bound (lb), and upper bound (ub) as input. ABC initializes a population of candidate solutions (bees) with random weights and biases.

Employed Bee Phase:
The algorithm proceeds through iterations and updates the candidate solutions in the employed bee phase. Each employed bee generates a new candidate solution by adjusting a random component of its weight and bias based on the information from other bees. If the updated solution improves the fitness (MSE), it replaces the current solution.

Scout Bee Phase:
In the scout bee phase, bees that have not found improved solutions are replaced with new random solutions within the specified bounds (lb and ub).

Convergence Visualization:
To assess the convergence behavior of the ABC algorithm, the code plots the error (MSE) over iterations for different population sizes (5, 10, 50, and 100). The analysis allows understanding how the population size affects the optimization process and the algorithm's overall performance.

Model Evaluation:
The code evaluates the best solution obtained through ABC by calculating the predicted stock prices using the optimal weights and bias. The mean squared error (MSE) between the predicted and actual stock prices on the training dataset is computed.

Results Visualization:
The predicted stock prices and the actual stock prices are plotted on the same graph to visually compare the model's performance.

Conclusion:
This academic report demonstrates the successful application of the Artificial Bee Colony (ABC) algorithm for stock price prediction using historical data of Apple Inc. The ABC algorithm effectively optimizes the model parameters, achieving reduced mean squared error (MSE) between the predicted and actual stock prices. The convergence visualization highlights the effect of different population sizes on the optimization process, offering valuable insights into the algorithm's performance.

It is important to note that stock price prediction is a complex task influenced by various dynamic factors. The results obtained through ABC should be thoroughly evaluated using appropriate validation techniques, such as cross-validation, to assess the model's generalization performance on unseen data.

Further research could explore the combination of ABC with other optimization algorithms or investigate advanced machine learning models for stock price prediction. Additionally, applying the ABC algorithm to other financial datasets and comparing its performance with alternative optimization methods could provide valuable knowledge for real-world financial decision-making.

Report done by:

Anish Devnoor
2020A8PS1344H

Aayush Mohanty
2020B3A71842