Computer Science and Engineering Department

Master

AI-CS-Group 16

Fitore Tahiri

February / 2025

Prishtinë

Contents

Introduction.. 3

N Queen Problem- Overview.. 4

Solution using Hill Climbing Algorithm.. 4

Learning: Classification.. 6

Training and testing the dataset.. 7

Conclusion 11

References.. 12

Introduction

This report represents the analysis and solutions of two AI problems: N Queen Problem and IRIS data classification using machine learning algorithms. For the N Queen Problem, the report will demonstrate a solution simulated using a Graphical User Interface in Python and explain the algorithms used for solving it.

Furthermore, will implement and evaluate classifiers using k-NN, Decision Tree, and Neural Network for the IRIS dataset. The evaluation process includes comparing these algorithms based on their accuracy, computational complexity, and suitability using the same dataset for all. Additionally, the report will discuss the strengths and weaknesses of each approach, highlighting scenarios where one algorithm may be preferable over the others.

For these tasks, environment used is: Python version 3.10.6, Windows 11 OS, and 8GB Ram memory. As of libraries used, main Python libraries have been mentioned inside the report.

N Queen Problem- Overview

Local search is an optimization strategy that seeks to find a good solution by gradually exploring possible configurations. These algorithms are used in large-scale problems where the search space is too vast for techniques like exhaustive search or dynamic programming. Some of the local search algorithms include:

1. **Hill Climbing:** This algorithm works by gradually improving an initial solution, selecting the move that reduces conflicts the most.
2. **Simulated Annealing:** Unlike Hill Climbing, Simulated Annealing allows occasional random moves to avoid getting stuck at local optima.
3. **Genetic Algorithms:** These algorithms generate a population of solutions and apply evolutionary operators like selection, crossover, and mutation to explore the search space.

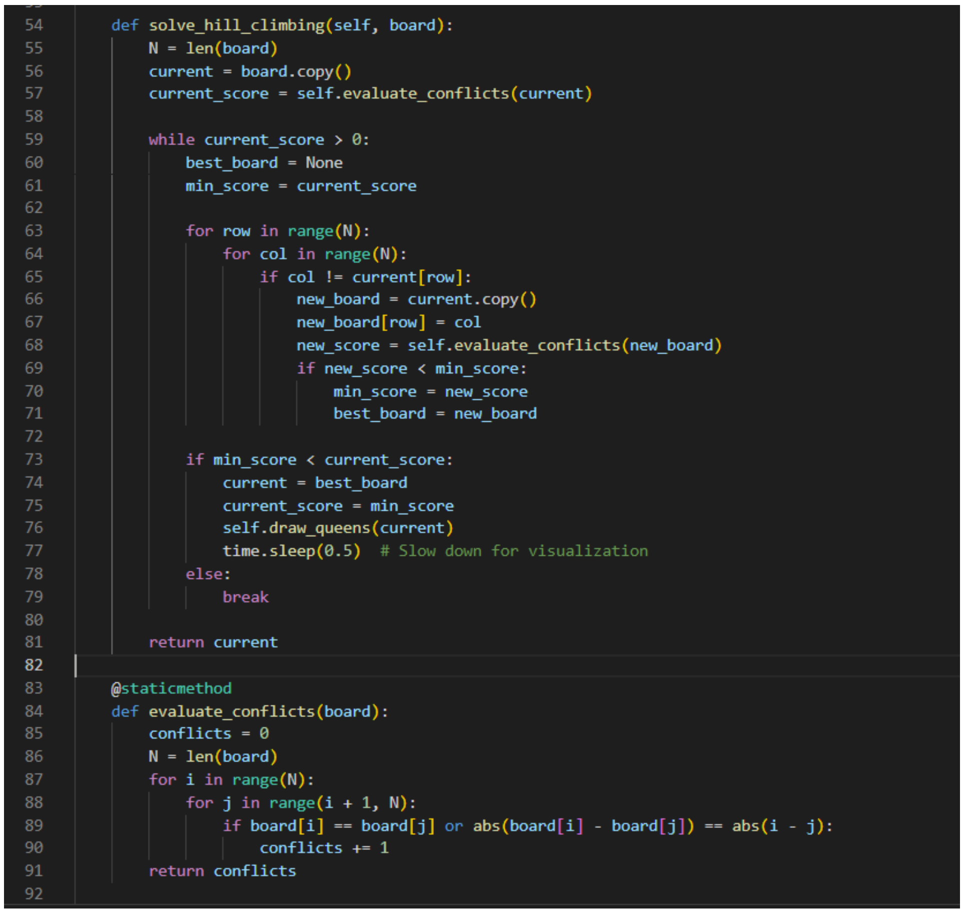
The goal of the N-Queen problem is to place N queens on an N × N chessboard such that no two queens attack each other. This means that no two queens can be in the same row, column, or diagonal.

Solution using Hill Climbing Algorithm

As already stated, Hill Climbing is a form of greedy algorithm that starts from an initial solution and makes incremental improvements to reach a goal. For example, in a problem of maximizing a function f(x), Hill Climbing would start at an initial point x0, evaluate the function f(x0), and then move to the neighboring point that gives the highest value of f(x). This continues until no better neighbor can be found.

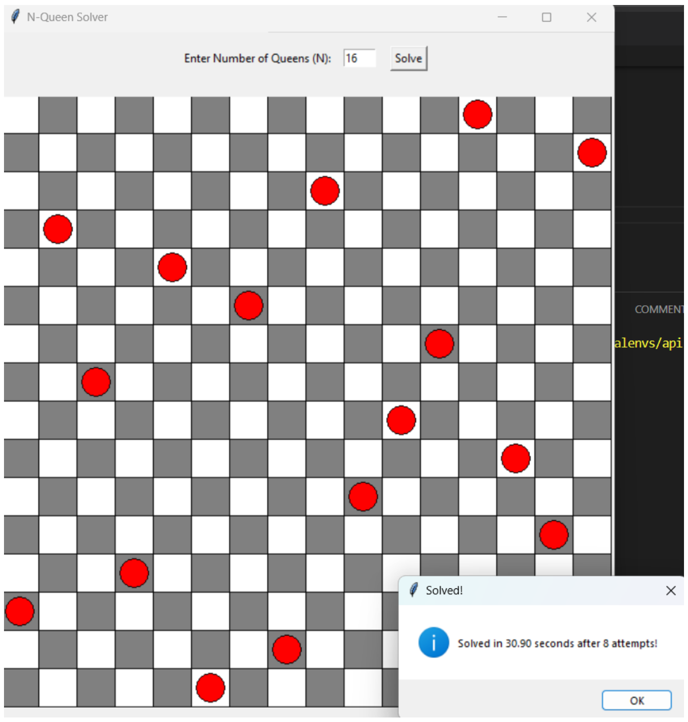
Advantages of this algorithm is that it is simple and easy to implement and it is also very efficient. But, sometimes it may not always find the optimal solution, especially in complex or large problems with many local optima.

For solving the N Queen Problem, Python has been used for implementing the algorithm and Python library tkinter has been used to implement the GUI, which visualizes the algorithm. The main fuction in the code is *solve_hill_climbing*. It tries to minimize the number of conflicts (attacks between queens) by iteratively adjusting the queen positions. The method begins by evaluating the board and looking for a neighboring board (new positions) with fewer conflicts. If the new configuration has fewer conflicts, the algorithm moves to that configuration and continues the process. This repeats until no more improvements can be made (i.e., the score reaches zero, meaning no conflicts). The *evaluate_conflicts* function counts the conflicts between queens (both row and diagonal attacks). Below is shown the parts of code and the GUI for after solving for N = 16:



```python
    def solve_hill_climbing(self, board):
        N = len(board)
        current = board.copy()
        current_score = self.evaluate_conflicts(current)

        while current_score > 0:
            best_board = None
            min_score = current_score

            for row in range(N):
                for col in range(N):
                    if col != current[row]:
                        new_board = current.copy()
                        new_board[row] = col
                        new_score = self.evaluate_conflicts(new_board)
                        if new_score < min_score:
                            min_score = new_score
                            best_board = new_board

            if min_score < current_score:
                current = best_board
                current_score = min_score
                self.draw_queens(current)
                time.sleep(0.5)  # Slow down for visualization
            else:
                break

        return current

    @staticmethod
    def evaluate_conflicts(board):
        conflicts = 0
        N = len(board)
        for i in range(N):
            for j in range(i + 1, N):
                if board[i] == board[j] or abs(board[i] - board[j]) == abs(i - j):
                    conflicts += 1
        return conflicts

```

Fig. 1. Main parts of the code using Hill Climbing Algorithm



Solved in 30.90 seconds after 8 attempts!

Fig. 2. Solution for N=16.

Learning: Classification

For this classification problem, IRIS dataset has been used. This dataset contains three different types of flowers: Setosa, Versicolor and Virginica, with four characteristics for each: length and width of petal and sepal.

For this problem, the following algorithms were used:

- **K-NN (k Nearest Neighbors):** The k-NN algorithm classifies a data point based on the majority class of its k nearest neighbors in the feature space. The distance between points is typically measured using Euclidean distance. This algorithm is easy to be implemented and has great performance for small datasets, but it is sensitive to irrelevant features and the choice of k.
- **Decision Tree Classifier:** A decision tree splits data into different regions based on feature values. This algorithms is easy to interpret (like a flowchart) which can be

understood even by non-tech people. But, it is prone to overfitting if not pruned (for example if it learns too much from training data and fails on new data). And another downside is that even small changes in data can lead to a completely different tree, which makes the algorithm unstable.

- **Neural Network:** Composed of layers of neurons: input layer → hidden layers → output layer. This algorithm uses weights that are adjusted using backpropagation to minimize classification error. This is a more complex algorithm, requires more computation power to train and works well with very large datasets. The more data you provide, the better the algorithm learns. Apart from it's complexity, another downside can be that it requires hyperparameter tuning.

Training and testing the dataset

As recommended, dataset was split into training 80%, and testing data 20%. The same data was fed to each algorithm. The main Python libraries used for training and testing the dataset are *sklearn* and *pandas*. Results after training for each classifier are shown in the table below:

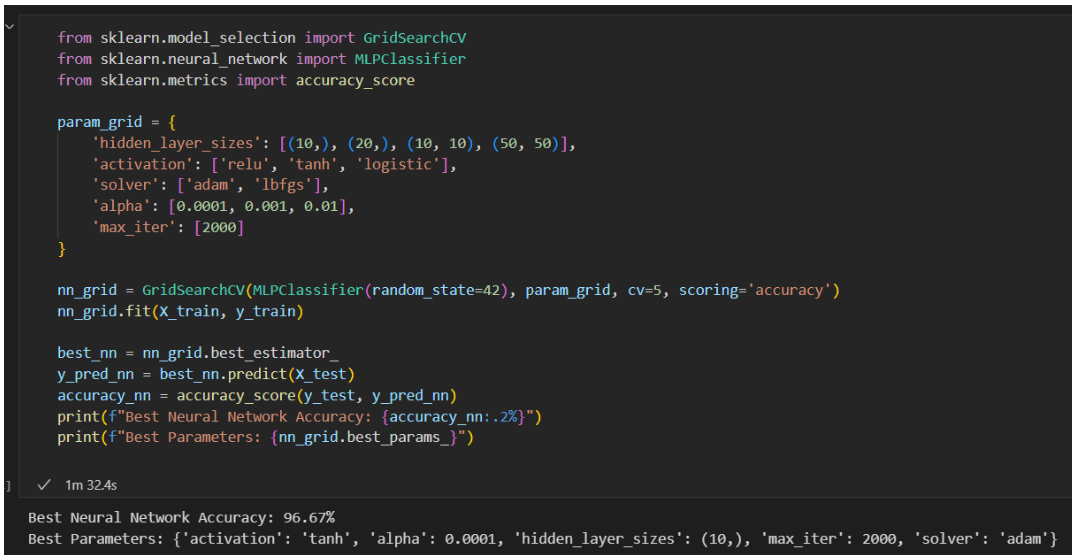
Algorithm	**Accuracy (%)**	**Time**
k-NN	93.33 %	0.0033395.. seconds
Decision Tree	90.00 %	0.004419... seconds
Neural Network	96.67 %	0.5571 seconds

As we can see, Neural Network gave the best results but execution time was longer while training. Meanwhile, Decision Tree and k-NN was faster but has a tendency to overfit.

Now will analyze each classifier how it was trained:

1. K-NN: used k=5. Meaning that the algorithm will consider the 5 nearest neighbors when making a prediction.
2. Decision Tree: the classifier uses a fixed random state (random_state=42). The random state ensures reproducibility of the results.
3. Neural Network: in *param_grid* dictionary, I have defined hyperparameters to be tested and performed a grid search over MLPClassifier (Multi-layer Perceptron) and the defined

hyperparameter grid. It uses 5-fold cross-validation (cv=5) and evaluates models based on accuracy. This means that for the provided hyperparameters, several Neural Network classifiers have been trained, and using grid search, will get the results of the most accurate model. The code is provided below, and as we can see the best parameters turned out to be: {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (10,), 'max_iter': 2000, 'solver': 'adam'}



```
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

param_grid = {
    'hidden_layer_sizes': [(10,), (20,), (10, 10), (50, 50)],
    'activation': ['relu', 'tanh', 'logistic'],
    'solver': ['adam', 'lbfgs'],
    'alpha': [0.0001, 0.001, 0.01],
    'max_iter': [2000]
}

nn_grid = GridSearchCV(MLPClassifier(random_state=42), param_grid, cv=5, scoring='accuracy')
nn_grid.fit(X_train, y_train)

best_nn = nn_grid.best_estimator_
y_pred_nn = best_nn.predict(X_test)
accuracy_nn = accuracy_score(y_test, y_pred_nn)
print(f"Best Neural Network Accuracy: {accuracy_nn:.2%}")
print(f"Best Parameters: {nn_grid.best_params_}")
```

✓ 1m 32.4s

```
Best Neural Network Accuracy: 96.67%
Best Parameters: {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (10,), 'max_iter': 2000, 'solver': 'adam'}
```

Fig. 3. Neural Network Classifier

Next, only petal length and petal width were selected as features from the dataset. And continued to test the same classifiers with the new data.

```
# Select only petal length and petal width as features
X_selected = df[['petallength', 'petalwidth']].values
X_train_sel, X_test_sel, y_train, y_test = train_test_split(X_selected, y,
test_size=0.2, random_state=42, stratify=y)
```

The new result is shown in the following table:

Algorithm	**Accuracy (%)**
k-NN	96.67 %
Decision Tree	93.33 %
Neural Network	96.67 %

As we can see, k-Nearest Neighbors and Decision Tree performed better with less features. k-NN and Decision Tree classifiers benefit from the reduced feature set because it simplifies the problem and focuses on the most relevant features. This process is also known as pruning, or removing irrelevant features in order to avoid overfitting. On the other hand, the reason why Neural Network performed the same is because Neural Networks are capable of learning complex patterns and relationships in the data. They can handle higher-dimensional data effectively. Removing features might not significantly impact their performance if the remaining features are still sufficient for accurate classification.

Finally, will present using confusion matices, the result of testing each classifier, which are conducted after removing the mentioned features from the data. In Fig.4, we can see that k-NN made mistake on two cases where Iris-virginica was labeled as Iris- versicolor. From Fig.5. Decision Tree made mistake on three cases by labeling Iris- versicolor as Iris-virginica and vice versa. While Neural Network (Fig.6) classifier made mistake in only one case by labeling Iris-versicolor as Iris-virginica.

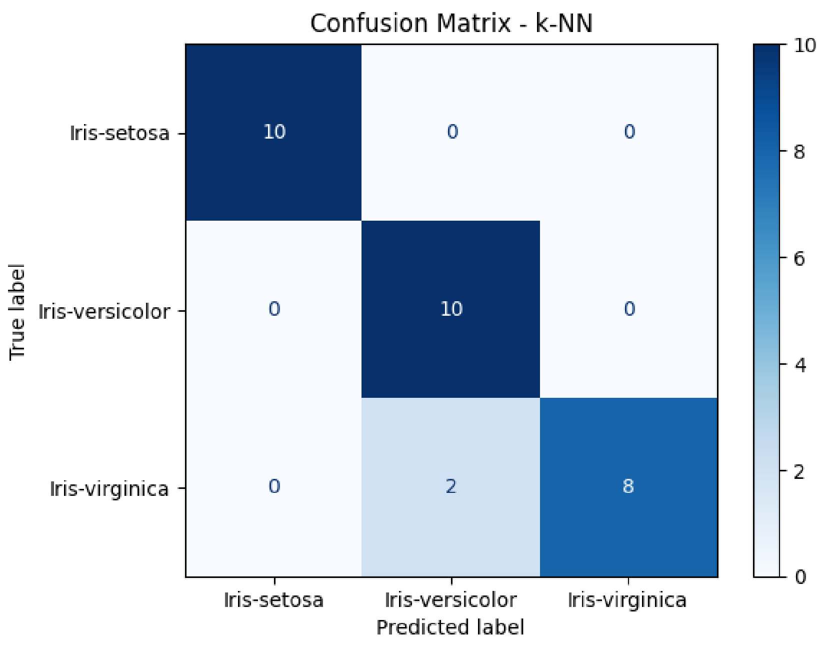


Fig.4. Confusion Matrix k-NN

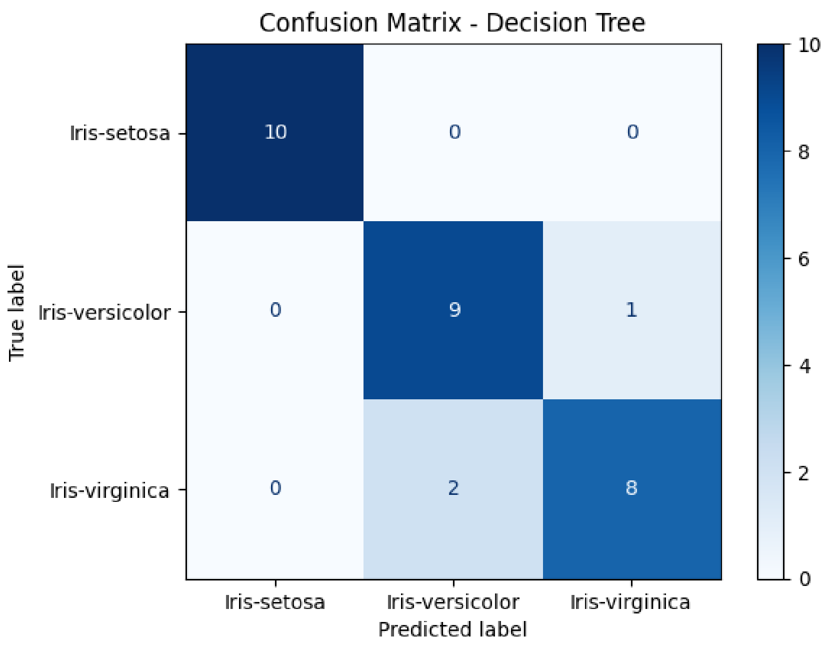


Fig.5. Confusion Matrix Decision Tree

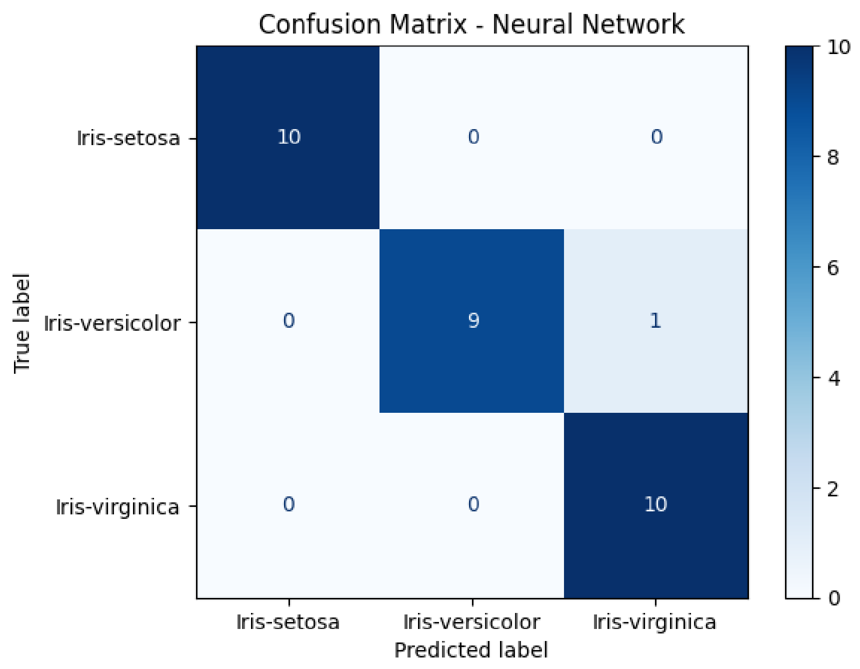


Fig.6. Confusion Matrix Neural Netwok

Conclusion

In this report two problems were handled: N Queen Problem using local search algorithms and also the classification problem. The Hill Climbing algorithm provides an effective yet simple approach to solving the N-Queen problem. While it is quick in many cases, the algorithm's susceptibility to local optima means that its performance can vary depending on the initial configuration. Further improvements, such as integrating techniques like Simulated Annealing or Genetic Algorithms, could enhance the algorithm's robustness, especially for larger values of N.

On the other hand, for the classification problem, the Neural Network classifier achieved competitive accuracy compared to k-NN and Decision Tree classifiers. Tuning hyperparameters such as hidden layer size and activation function significantly impacted performance. While k-NN and Decision Tree models are easier to interpret, NN provides flexibility and generalization ability, making it a strong choice for classification tasks.

References

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer. Retrieved from https://www.springer.com/gp/book/9780387310732

Russell, S., & Norvig, P. (2016). *Artificial Intelligence: A Modern Approach* (3rd ed.). Pearson Education. Retrieved from https://www.pearson.com/store/p/artificial-intelligence-a-modern-approach/P100000592736

GeeksforGeeks. (2021). *N-Queen Problem | Local Search using Hill Climbing with Random Neighbour*. Retrieved from https://www.geeksforgeeks.org/n-queen-problem-local-search-using-hill-climbing-with-random-neighbour/

KeyLabs. (2021). *Neural Networks and Deep Learning for Classification*. Retrieved from https://keylabs.ai/blog/neural-networks-and-deep-learning-for-classification/