Homework 04

[2024-1] 데이터사이언스를 위한 컴퓨팅1 (001)

Due: 2024**년 **6**월 **16**일 **23:59

In this homework, you will solve problems using the various algorithms presented in the class

1. Shortest Path between all location[25pts]

Read the below instructions and the given source code hw4_1.hpp and main.cpp properly and follow it.

- Implement the void initializeGraph. This function should initialize the distance and predecessor matrix before the void DistanceMatrix function.
- Implement the void DistanceMatrix function. This function should get the initialized distance, predecessor matrix as input and fill the distance matrix with shortest path length within each location and fill the predecessor matrix with the predecessor of each node in the shortest path. The time complexity of this function should be O(n^3).
- Implement the void printPath function that prints the shortest path from start to end. The inputs of this function are the predecessor matrix, start index, end index, and location_list(actual name of the location) and print the actual name of the location. Printing order should be “{start}-{node}-{node}- … -{end}”. If the path doesn’t exist, it should print “No Path”.
- Implement the void NegCycleDetection. This function should print “negative” if the graph has a negative weight cycle; otherwise, it should print “no negative”.
- Use the infinite number as a predefined const int INF(999) in the distance matrix, and NIL as -1 in the predecessor matrix.

The grading will be based on the result of the printPath, NegCycleDetection function and the running time of DistanceMatrix function.

Example:

```
int n=5,start=0,end=3;
vector<string> location_list = {"A","B","C","D","E"};
vector<vector<int>> distance(n, vector<int>(n, INF)); //INF=999
vector<vector<int>> predecessor(n, vector<int>(n));
vector<vector<int>> vertex_edge = {
    {0,1,3},
    {0,2,8},
    {0,4,-4},
    {1,3,1},
    {1,4,7},
    {2,1,4},
    {3,0,2},
    {3,2,-5},
    {4,3,6}
};
initializeGraph(distance, predecessor, vertex_edge);
```

```
DistanceMatrix(distance, predecessor);
printPath(predecessor, start, end, location_list);
cout << endl;
NegCycleDetection(distance);

>>> A-E-D-
>>> no negative
```

2. Balanced Tree[25pts]

Read the below instructions and the given source code hw4_2.hpp and main.cpp properly and follow it.

- Implement the void RBTree::fixInsertRBTree function.
- During the implementation, use the predefined void RBTree::rotateRight, void RBTree::rotateLeft function.

The grading will be based on the printed result of inorder traverse and running time of predefined void RBTree::search function.

Example:

```
RBTree tree;

tree.insert(2);
tree.insert(4);
tree.insert(3);
tree.insert(7);
tree.insert(8);
tree.insert(11);
tree.insert(1);

tree.inorder(); // This is predefined in hpp source code.
cout << endl;

tree.search(11); // This is predefined in hpp source code.

>>> 1-2-3-4-7-8-11-
>>> 3
```

3. KNN[25pts]

Read the below instructions and the given source code hw4_3.hpp and main.cpp properly and follow it.

- Implement the int KNN::infer function. This function performs the inference algorithm of K nearest neighbor method. During implementation, you can use the void sort function from the algorithm library. Additionally, implement tie-breaking in the infer function of KNN by randomly selecting one class among the tied classes when there are the same number of instances for multiple classes among the K nearest neighbors.
- Implement the two distance metric, double KNN::euclideanDistance and double KNN::cosineSimilarity. During implementation, using sqrt from cmath library and inner_product from numeric library is recommended.

The grading will be based on the result of int KNN::infer, double KNN::euclideanDistance, double KNN::cosineSimilarity.

Example:

```
// Read the dataset (excluding the last column for features)
vector<vector<double>> dataset = readCSV("iris.csv");
vector<vector<double>> X;
vector<int> y;

for (const auto& row : dataset) {
    vector<double> features(row.begin(), row.end() - 1);
    int label = static_cast<int>(row.back());
    X.push_back(features);
    y.push_back(label);
}
// Create object & training
// arg1: k of knn, arg2: “E” for euclidean dist, “C” for cos sim
KNN knn(3, "E");

knn.train(X, y);

// Test one example from the dataset
vector<double> test_sample = {5.1,3.5,1.4,0.2};
int predicted_label = knn.infer(test_sample);

cout << "Predicted label: " << predicted_label << endl;

>>> Predicted label: 0
```

4. Gradient Descent[25pts]

Read the below instructions and the given source code hw4_4.hpp and main.cpp properly and follow it.

- Implement the vector<double> gradientDescent function. This function is capable of training a multi-linear regression model and should return the trained model's parameters as a vector. Keep in mind that the bias corresponds to the last value of the weight vector. By examining the example dataset, you can confirm that the rightmost column of the matrix is filled entirely with 1s.
- To check the convergence during training, you can use a predefined loss function inside the gradientDescent function.

The grading will be based on the Mean Squared Error of the test dataset and the running time.

Example:

```
// data load & train test split
vector<vector<double>> X = readCSV("data.csv");
vector<double> y = readLabels("label.csv");

int n_train = X.size()*0.8;
int n_test = X.size()-n_train;

vector<vector<double>> X_train;
vector<double> y_train;
for(int i=0;i<n_train;i++){
    X_train.push_back(X[i]);
    y_train.push_back(y[i]);
}
vector<vector<double>> X_test;
vector<double> y_test;
for(int i=n_train;i<X.size();i++){
    X_test.push_back(X[i]);
    y_test.push_back(y[i]);
}

vector<double> w(X[0].size(), 0.0);
double alpha = 0.01;
int num_iters = 1000;

// train & infer
w = gradientDescent(X_train, y_train, w, alpha, num_iters);

double mse = 0;
for(int i=0;i<n_test;i++){
    double pred=0;
    for(int j=0;j<X[0].size();j++){
        pred+=w[j]*X_test[i][j];
    }
    mse+=pow(y_test[i]-pred,2);
}
```

```
cout << "MSE:" << mse/n_test << endl;

>>> MSE:0.581472
```

5. Submission Instructions

- Follow the instructions of each problem and fill each “hw4_{n}.hpp” files. You can modify other files for your own testing, but keep in mind that **only “hw4_n.hpp” will be graded using the original header files.**
- Grading will be conducted using the C++11 standard. Our compilation process is as follows:
  - `$ g++ main.cpp -o main -std=c++11`
- You should NOT share your code with other students. Any student found to have a high level of code similarity, as determined by our similarity detection process, may face severe penalties.
- In this assignment, Grace day is not allowed. The deadline is June 16th at 23:59. If you do not submit the assignment by then, you will not receive any points.
- For questions about this homework, use the “Homework 4” discussion forum on eTL.
- Failing to adhere to the submission guidelines may result in penalties applied without exception.