Lab04

CSV 파일 읽기

```
int main()
{
  // CVS 파일 읽어오기
  std::string CsvPath = std::string("..\\Run\\diabetes.csv");
  // 실행될 창 생성
  CPerformance *pPerformance = new CPerformance(600, 600, CsvPath);
```

```
CPerformance::CPerformance(int nW, int nH, std::string CsvPath) : CKhuGleWin(nW, nH)
{
  m_pScene = new CKhuGleScene(520, 530, KG_COLOR_24_RGB(100, 100, 150));

  m_pRocLayer = new CKhuGleRocLayer(480, 480, KG_COLOR_24_RGB(150, 150, 200), CKgPoint(20, 30));
  m_pScene->AddChild(m_pRocLayer);

  // csv 파일을 읽어오고 해당 정보를 저장함
  ReadCsv(CsvPath, m_pRocLayer->m_Head, m_pRocLayer->m_ReadData);
  // , 기준으로 데이터가 나뉘어져 있기에 \n과 ,을 기준으로 데이터를 추출하게됨
  for (auto& column : m_pRocLayer->m_Head) {
    std::cout << column << ", "; // 해당 정보 출력
  }
  std::cout << std::endl;
```

CSV 파일 데이터 가공 및 정리

```
// z-score 정규화(normalization) 진행
  // z-score 정규화란?: 데이터를 평균 0, 표준 편차 1로 변환
  // 먼저 평균과 표준 편차 계산
  for (int k = 0; k < mean.size(); ++k) {
    mean[k] /= m_pRocLayer->m_ReadData.size();

    // 표준 편차는 분산의 제곱근
    // 분산은 각 데이터 포인트와 평균의 차이의 제곱의 평균
    sd[k] = sd[k] / m_pRocLayer->m_ReadData.size() - mean[k] * mean[k];
    sd[k] = sqrt(sd[k]);
  }

  // 정규화된 데이터를 출력합니다.
  int nPrintCnt = 0;
  for (auto& read : m_pRocLayer->m_ReadData) {
    for (auto& column : read) {
      std::cout << column << ", ";
    }
    std::cout << std::endl;

    if (nPrintCnt++ > 10) break;
  }

  for (auto& read : m_pRocLayer->m_ReadData) {
    for (int k = 0; k < mean.size(); ++k) {
      read[k] = (read[k] - mean[k]) / sd[k];
    }
  }

  // z-score 정규화된 데이터를 출력
  std::cout << "\nZ-socre normalization" << std::endl;
  nPrintCnt = 0;
  for (auto& read : m_pRocLayer->m_ReadData) {
    for (auto& column : read) {
      std::cout << column << ", ";
    }
    std::cout << std::endl;

    if (nPrintCnt++ > 10) {
      break;
```

```
    }
  }
```

학습 데이터 및 실험 데이터 설정

```
// 학습하는 데이터와 실제 사용하는 데이터 구분
std::vector<bool> Train(m_ReadData.size());
int nTrain = 0, nTest = 0;
for(int i=0; i<m_ReadData.size(); i++){
  if (rand() % 5 < 4) {
    Train[i] = true;
    nTrain++;
  }
  else {
    Train[i] = false;
    nTest++;
  }
}
std::cout << "\n*********************************" << std::endl;
std::cout << "Total data: " << m_ReadData.size() << std::endl;
std::cout << "Train data: " << nTrain << std::endl;
std::cout << "Test data: " << nTest << std::end
```

데이터 분석

```
// 2 ~ 15 까지의 수에서 K값을 선정 / 이외의 값들 초기화
int nk = rand() % 14 + 1; // 2부터 15까지의 랜덤한 K값을 선택
double nTP = 0, nFP = 0, nTN = 0; // True Positive(TP), False Positive(FP), True Negative(TN)의 초기값을 0으로 설정
double positive = 0, negative = 0; // Positive와 Negative의 초기값을 0으로 설정

for (int i = 0; i < m_ReadData.size(); i++) {
  // 학습 데이터면 실험
  if (Train[i]) {
  continue; // 학습 데이터인 경우 루프의 다음 반복으로 건너뛰고, 다음 데이터로 이동
  }

  std::vector<std::pair<double, int>> NN; // NN은 이웃 데이터를 저장하는 벡터
  for (int j = 0; j < m_ReadData.size(); j++) {
    double dist = 0; // 거리를 계산하기 위한 변수 dist 초기화
    if (!Train[j]) {
      continue; // 학습 데이터가 아닌 경우 루프의 다음 반복으로 건너뛰고, 다음 데이터로 이동
    }
    for (int k = 0; k < m_ReadData[j].size() - 1; k++) {
      dist += (m_ReadData[i][k] - m_ReadData[j][k]) * (m_ReadData[i][k] - m_ReadData[j][k]); // 유클리드 거리 계산
    }
    dist = sqrt(dist); // 거리값의 제곱근을 구함
    NN.push_back({ dist, (int)m_ReadData[j][m_ReadData[j].size() - 1]}); // 이웃 데이터의 거리와 클래스 값을 NN 벡터에 추가
  }
  std::sort(NN.begin(), NN.end(), DistanceSort); // NN 벡터를 거리 기준으로 정렬

  int decision = 0; // 분류 결정을 위한 변수 초기화

  for (int j = 0; j < nk; j++) {
    decision += NN[j].second; // 가장 가까운 K개의 이웃 데이터의 클래스 값을 더함
  }

  m_Data.push_back({ (int)m_ReadData[i][m_ReadData[i].size() - 1], (double)decision / (double)nk }); // 분류 결과를 m_Data에 추가

  if (decision > nk / 2.0)
    decision = 1; // 분류 결정이 K/2보다 큰 경우 1로 설정
  else
    decision = 0; // 그렇지 않은 경우 0으로 설정

  if (decision == 1)
  {
    int real = (int)m_ReadData[i][m_ReadData[i].size() - 1]; // 실제 클래스 값
    if (real == 1)
      nTP++; // TP 증가
    else
      nFP++; // FP 증가
```

```
      positive++; // Positive 증가
    }
    else {
      int real = (int)m_ReadData[i][m_ReadData[i].size() - 1]; // 실제 클래스 값
      if (real == 0)
        nTN++; // TN 증가
      negative++; // Negative 증가
    }
  }
```

결론

자료를 분석하고 정리하는 과정에서 최대값과 최소값에 대해서 범위를 정해서 분석하는 것이 아닌 z-score로 분석하는 과정은 더 효과적이고 실효적인 데이터 분석이 될 수 있다. z-score는 데이터의 각 특성을 평균과 표준 편차를 이용하여 Z-score로 결론을 낸다. 이는 데이터의 분포를 평균이 0이고 표준 편차가 1인 정규분포로 조정하는 과정으로, 다양한 특성을 동일한 척도로 맞추는 데 도움을 주게 된다.

True Positive Rate(TPR), False Positive Rate(FPR), Accuracy 등의 성능 지표를 계산하여 모델의 예측 성능을 평가하는 과정을 거친 데이터를 통해서 k-NN 알고리즘을 실행한다.k 값은 임의로 설정되며, [2,15] 범위에서 선택된다. 결론적으로 데이터가 잘 추출되고 잘 실행된다.

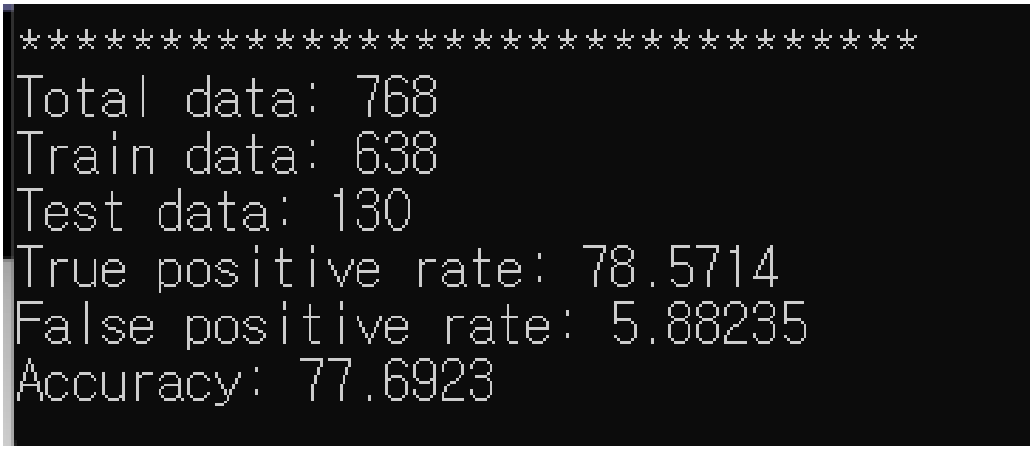
(R: 다시 실행, Q: 프로세스 종료)

데이터 추출 및 z 정규화

```
Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age, Outcome,
6, 148, 72, 35, 0, 33.6, 0.627, 50, 1,
1, 85, 66, 29, 0, 26.6, 0.351, 31, 0,
8, 183, 64, 0, 0, 23.3, 0.672, 32, 1,
1, 89, 66, 23, 94, 28.1, 0.167, 21, 0,
0, 137, 40, 35, 168, 43.1, 2.288, 33, 1,
5, 116, 74, 0, 0, 25.6, 0.201, 30, 0,
3, 78, 50, 32, 88, 31, 0.248, 26, 1,
10, 115, 0, 0, 0, 35.3, 0.134, 29, 0,
2, 197, 70, 45, 543, 30.5, 0.158, 53, 1,
8, 125, 96, 0, 0, 0, 0.232, 54, 1,
4, 110, 92, 0, 0, 37.6, 0.191, 30, 0,
10, 168, 74, 0, 0, 38, 0.537, 34, 1,

Z-socre normalization
0.639947, 0.848324, 0.149641, 0.90727, -0.692891, 0.204013, 0.468492, 1.426, 1,
-0.844885, -1.1234, -0.160546, 0.530902, -0.692891, -0.684422, -0.365061, -0.190672, 0,
1.23388, 1.94372, -0.263941, -1.28821, -0.692891, -1.10326, 0.604397, -0.105584, 1,
-0.844885, -0.998208, -0.160546, 0.154533, 0.123302, -0.494043, -0.920763, -1.04155, 0,
-1.14185, 0.504055, -1.50469, 0.90727, 0.765836, 1.40975, 5.48491, -0.0204964, 1,
0.342981, -0.153185, 0.253036, -1.28821, -0.692891, -0.811341, -0.818079, -0.27576, 0,
-0.250952, -1.34248, -0.98771, 0.719086, 0.0712043, -0.125977, -0.676133, -0.616111, 1,
1.82781, -0.184482, -3.5726, -1.28821, -0.692891, 0.419775, -1.02043, -0.360847, 0,
-0.547919, 2.38188, 0.0462453, 1.53455, 4.02192, -0.189437, -0.947944, 1.68126, 1,
1.23388, 0.128489, 1.39039, -1.28821, -0.692891, -4.06047, -0.724455, 1.76635, 1,
0.0460143, -0.340968, 1.1836, -1.28821, -0.692891, 0.71169, -0.84828, -0.27576, 0,
1.82781, 1.47427, 0.253036, -1.28821, -0.692891, 0.762457, 0.196681, 0.0645914, 1,
```

데이터 분석


****
Total data: 768
Train data: 638
Test data: 130
True positive rate: 78.5714
False positive rate: 5.88235
Accuracy: 77.6923