

**EGE ÜNİVERSİTESİ**

**MÜHENDİSLİK FAKÜLTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

**YAPAY ZEKÂ YÖNTEMLERİ (3+0)**

**2022-2023 BAHAR YARIYILI**

**PROJE-1 RAPORU**

**TESLİM TARİHİ**

--/--/---

**HAZIRLAYANLAR**

05190000114 – Mahmut Çelik

05190000027 – Özgür Bayraşa

05220000011 – Sedat Korkmaz

İçindekiler

[1) 1) Algoritmalar, Tanımlar, Karşılaştırma, Araştırma ve Yorum 2](#_Toc100068189)

[1.a Kalemle yazılan algoritmaların tarayıcı görüntüleri veya fotoğrafları 2](#_Toc100068190)

[1.b Tanım ve Karşılaştırmalar 2](#_Toc100068191)

[1.c Araştırma ve Yorum 2](#_Toc100068192)

[2) Problem Çözme ve Kodlama 2](#_Toc100068193)

[2.a Problemin Tanımı 2](#_Toc100068194)

[2.b Programın Özellikleri 2](#_Toc100068195)

[2.c Programın Ekran Görüntüleri 2](#_Toc100068196)

[2.d Çözüm Başarısı/Çözüm Adımları 2](#_Toc100068197)

[3) Genetik Algoritmalar ile Şifre Kırma 2](#_Toc100068198)

[3.a İlgili maddede istenenler ve karşılaştırma (kromozom sayısının etkisi) 2](#_Toc100068199)

[3.b Kod; Çaprazlama ve Mutasyon Fonksiyonlarının Anlatımı 2](#_Toc100068200)

[4) Makine Öğrenmesi 2](#_Toc100068201)

[4.a Verisetinin ve problemin kısa anlatımı 2](#_Toc100068202)

[4.b İki Farklı Sınıflandırıcı için Sonuçlar: Hata Matrisleri, Tablo 2](#_Toc100068203)

[4.c Kullanıcı tarafından verilen örneğin sınıflandırma ekran görüntüsü (konsol çıktıları) 2](#_Toc100068204)

[4.d Kod 2](#_Toc100068205)

[5) Öz değerlendirme Tablosu 2](#_Toc100068206)

# 1) 1) Algoritmalar, Tanımlar, Karşılaştırma, Araştırma ve Yorum

## 1.a Kalemle yazılan algoritmaların tarayıcı görüntüleri veya fotoğrafları

//Mürekkepli (veya kurşun) kalemle bir kâğıda yazınız.

## 1.b Tanım ve Karşılaştırmalar

//Bilgisayarda yazılabilir

## 1.c Araştırma ve Yorum

//Mürekkepli (veya kurşun) kalemle bir kâğıda yazınız. Bilgisayarda yazanlar yarım puan alacak.

# 2) Problem Çözme ve Kodlama

## 2.a Problemin Tanımı

Eight Queens problemi, 8 Vezir bulmacası 8\*8 satranç tahtası üzerinde 8 vezirin birbirini yemeyecek şekilde konumlandırılmasıdır. ( Aynı satır, aynı sütun veya aynı çarpraz sırada iki veya fazla vezir bulunamaz.)

## 2.b Çözüm Mekanizması ve Kaynak Kod

Çözüm mekanizması olarak, başlangıçta tüm vezirler array de 0 ile 7 arasında rastgele yerleştirilir. Başlangıç tahtasının çözülmüş bir şekilde olup olmadığı kontrol edilir. Kontrol için çapraz yeme ve yatay yeme durumlarına dikkat edilir. Herhangi bir yeme durumu olmuyorsa tahta çözülmüştür. Tahta çözülmemişse mevcut tahtanın neighbor state leri bulunur ve map içerisine yerleştirilir. Yerleştirilirken neighbor state lerdeki yeme durumları da hesaplanarak yerleştirilir. Yerleştirme sonrası en iyi durumdaki neighbor state e geçiş yapılır. Birden fazla en iyi durumda neighbor state varsa bu durumda random olarak birine geçilir. Neighbor state e geçilmeden önce kontrol mekanizması bulunuyor. Bu kontrolde şuanki state le geçilebilecek en iyi neighbor state teki yeme durumları karşılaştırılıyor. Değerde iyileşme olmuyorsa local min e takılmış oluyoruz ve random restart la tahtayı rastgele tekrar dizip baştan başlıyoruz. Değerde iyileşme varsa yeni state e geçilir ve bu şekilde ilerleyerek solution a ulaşırız.

*package* org.example;  
*public class* Main {  
 *//Çözüm yolu olarak Map kullanmayı tercih ettik. Alternatif olarak temporary array de kullanabilirdik.  
 //Aybars hocaya sorduğumuzda Map şeklinde kalabileceğini ve size de iletmemizi söyledi  
 public static void* main(String[] args) {  
 EightQueens queens = *new* EightQueens();  
 queens.solveNineTimes();  
 }  
}

*package* org.example;  
  
*import* java.util.\*;  
*import* java.util.concurrent.atomic.AtomicInteger;  
  
*public class* EightQueens {  
 *private static* Integer[] board = *new* Integer[8]; *// Chess Board -> Elements are columns and Indexes are rows  
 private static final* Map<Integer[], Integer> neighborStates = *new* LinkedHashMap<>();  
 *// Neighbor states of current board -> It holds neighbor state and number of eating each other count  
 private final* Random random = *new* Random();  
  
 *private static final* Object[][] solutionTable = *new* Object[9][5]; *//Keeps our solutions results  
  
 private static int* randomRestartCount = 0; *//Counts random restart for each solution tour  
  
 private static* Replacement replacement = *new* Replacement(*new* ArrayList<>(), 0);  
 */\*  
 \* Replacement has two fields  
 \* 1. field is a list that holds replacement counts for each random restart  
 \* 2. field is an int that holds current replacement count for that board  
 \* \*/  
  
 /\*\*  
 \* It calls solve() nine times and sets values in each loop  
 \* To calculate time -> System.nanoTime()  
 \* Finally prints the solutions to the console  
 \*/  
 public void* solveNineTimes() {  
  
 *for* (*int* x = 0; x < 9; x++) {  
 *double* time = 0;  
 *long* startTime = System.nanoTime();  
  
 replacement.setCurrentReplacementCount(0); *// Set 0 replacement - beacuse new board will start* replacement.setReplacementCounts(*new* ArrayList<>()); *//Refresh the replacement list beacyse new tour will start* randomRestartCount = 0; *// Set 0 random restart - beacuse new board will start* solve();  
  
 *long* endTime = System.nanoTime();  
  
 time = (endTime - startTime) / 1000000000.0;  
  
 solutionTable[x][0] = replacement.getSumOfReplacements(); *//Fill the table* solutionTable[x][1] = (*double*) randomRestartCount;  
 solutionTable[x][2] = time;  
 solutionTable[x][3] = replacement.getReplacementCounts();  
 solutionTable[x][4] = board.clone();  
  
 }  
 printSolutionTable();  
 }  
  
 */\*\*  
 \* 1 -> Generate random chess table via randomReplacement() method  
 \* 2 -> While loop continues until a solution is found  
 \* 2.1 -> If program encountered a local min, method calls itself again  
 \* (Current state value -> 5 , neighbor states values are 5 or bigger - It means local min)  
 \* 3 -> If everthing is okey, State will change via changeState()  
 \* (Current state value -> 5 , neighbor states contain a number that lower than 5)  
 \*/  
 public void* solve() {  
 randomPlacement();  
 *while* (numOfEating(board) != 0) { *//RANDOM RESTART YAPTIKTAN SONRA ONCEKI TURDAKI REPLACEMENT LARI DA SAYMALI MIYIZ?* replacement.increaseFinalReplacement();  
 fillNeighbors();  
 *if* (!neighborStates.values().stream().anyMatch(e -> e < numOfEating(board))) {  
 replacement.stateChange();  
 randomRestartCount++;  
 solve();  
 *break*;  
 }  
 changeState();  
 }  
 }  
  
 */\*\*  
 \* Randomly places queens  
 \*/  
 public void* randomPlacement() {  
 *for* (*int* x = 0; x < board.length; x++) {  
 board[x] = random.nextInt(0, 8);  
 }  
 }  
  
 */\*\*  
 \* It has one parameter and it calculates the parameter's eating count  
 \* 1 - Cross eating control  
 \* 2 - Horizontal eating control  
 \*/  
 public* Integer numOfEating(Integer[] val) {  
 *int* count = 0;  
 *for* (*int* x = 0; x < val.length - 1; x++) {  
 *for* (*int* y = x + 1; y < val.length; y++) {  
 *if* ((Math.abs(val[y] - val[x])) == (Math.abs(y - x)) || val[x] - val[y] == 0) {  
 count++;  
 }  
 }  
 }  
 *return* count;  
 }  
  
 */\*\*  
 \* Calculates 56 neighbor states of current state  
 \*/  
 public void* fillNeighbors() {  
 neighborStates.clear();  
 *for* (*int* x = 0; x < board.length; x++) {  
 Integer[] copyBoard = board.clone();  
 *for* (*int* y = 0; y < board.length; y++) {  
 *if* (y != board[x]) {  
 copyBoard[x] = y;  
 neighborStates.put(copyBoard.clone(), numOfEating(copyBoard));  
 }  
 }  
 }  
 }  
  
 */\*\*  
 \* It changes the current state to neighbor state that has min number value  
 \* ! -> If there are more than 1 state that has min number value, program will choose next state randomly  
 \*/  
 public void* changeState() {  
 *int* minNumber = findMin();  
 List<Map.Entry<Integer[], Integer>> arr = neighborStates  
 .entrySet()  
 .stream()  
 .filter(e -> e.getValue() == minNumber)  
 .toList();  
  
 board = arr.get(random.nextInt(arr.size())).getKey().clone();  
 }  
  
 */\*\*  
 \* Finds the min number of eating in neighbor states values  
 \*/  
 private* Integer findMin() {  
 *return* neighborStates  
 .values()  
 .stream()  
 .min(Integer::compare)  
 .orElseThrow(NullPointerException::*new*);  
 }  
  
 */\*\*  
 \* Printing solution table  
 \*/  
 public void* printSolutionTable() {  
 System.out.printf("-------------------------------------------------------------------------------------%n");  
 System.out.printf("| %-30s | %-30s | %15s |%n", "Replacement Count", "Random Restart Count", "Time");  
 System.out.printf("-------------------------------------------------------------------------------------%n");  
 *for* (*int* y = 0; y < solutionTable.length; y++) {  
 System.out.printf("| %-30s | %-30s | %15s |%n", solutionTable[y][0], solutionTable[y][1], solutionTable[y][2]);  
 }  
 System.out.printf("-------------------------------------------------------------------------------------%n");  
 System.out.println("| Replacement counts per Loop");  
 *for* (*int* z = 0; z < solutionTable.length; z++) {  
 System.out.println("| TOUR " + (z + 1) + ": " + solutionTable[z][3].toString());  
 }  
 System.out.printf("-------------------------------------------------------------------------------------%n");  
 System.out.println("| Final boards per Loop");  
 *for* (*int* z = 0; z < solutionTable.length; z++) {  
 System.out.println("| TOUR " + (z + 1) + ": " + Arrays.toString((Integer[])solutionTable[z][4]));  
 }  
 System.out.printf("-------------------------------------------------------------------------------------%n");  
  
 }  
}  
  
*/\*\*  
 \* Below class has 2 fields to hold replacement count for table - To provide each tours replacement count  
 \*/  
class* Replacement {  
 List<Integer> replacementCounts;  
 Integer currentReplacementCount;  
  
 *public* Replacement(List<Integer> replacementCounts, Integer currentReplacementCount) {  
 *this*.replacementCounts = replacementCounts;  
 *this*.currentReplacementCount = currentReplacementCount;  
 }  
  
 *public* List<Integer> getReplacementCounts() {  
 *return* replacementCounts;  
 }  
  
 *public void* setReplacementCounts(List<Integer> replacementCounts) {  
 *this*.replacementCounts = replacementCounts;  
 }  
  
 *public void* setCurrentReplacementCount(Integer currentReplacementCount) {  
 *this*.currentReplacementCount = currentReplacementCount;  
 }  
  
 *//Increases number of replacement of current tour  
 public void* increaseFinalReplacement() {  
 *this*.currentReplacementCount++;  
 }  
  
 *//Calculates the final replacement count of tour  
 public int* getSumOfReplacements() {  
 AtomicInteger sum = *new* AtomicInteger();  
 replacementCounts.forEach(sum::addAndGet);  
 *return* sum.get();  
 }  
  
 *//While state changes, current replacement count will add to the list and set current rep. to 0  
 public void* stateChange() {  
 replacementCounts.add(currentReplacementCount);  
 currentReplacementCount = 0;  
 }  
}

## 2.c Programın Ekran Görüntüleri

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

## 2.d Sonuç Tablosu

metin içeren bir resim

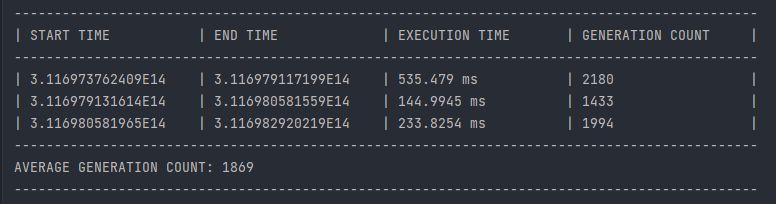
Açıklama otomatik olarak oluşturuldu

# 3) Genetik Algoritmalar ile Şifre Kırma

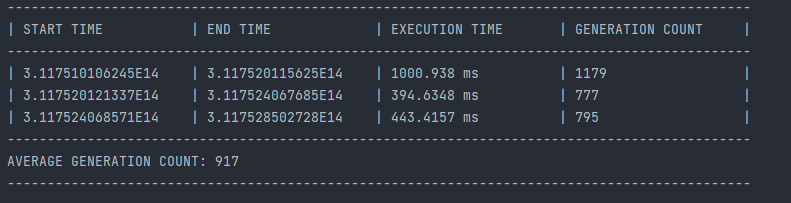
## 3.a İlgili Maddede İstenenler ve Karşılaştırma (kromozom sayısının etkisi)

Chromosome sayısının artması execution time da belirgin bir artışa sebep olmaktadır çünkü generation ları oluşturmak daha uzun sürecektir. Chromosome sayısındaki artış şifrenin kaç generation da bulunduğu sayısında düşüşü de sağlamaktadır. Daha fazla chromosome oluşturarak daha iyi genlere sahip chromosome elde etme şansımız ve asci aralığındaki değerlerin çoğuna sahip olmamızı sağlar.

20 Chromosome daki sonuçlar



100 Chromosome daki sonuçlar



1000 Chromosome daki sonuçlar

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

## 3.b Kod; Çaprazlama ve Mutasyon Fonksiyonlarının Anlatımı

Seçilim: Seçilim için derste anlatıldığı gibi roulettWheel yöntemini kullandım. Generationdaki her bir chromosome un fitness değerini 1/n şeklinde toplayıp bunu yüzdelik hale çevirdim bu sayede her bir fitness değerinin yüzdelik miktarını hesaplamak için çarpılması gereken sayıyı buluyoruz. Sonrasında bir max değerden random sayı ürettim (Başlangıçta 100 sonrasında azalıyor) ve generation u dolaşarak başlangıçta hesapladığım çarpma değerini de kullanarak random oluşturduğum sayıdan çıkararak ilerledim ve negatife düşmediği sürece devam etti

(Başlangıçta 100 random 20 geldi, 1.Chromosome sonucunda random sayı 15 e düştü, 2.chromsome sonucunda random sayı 9 a düştü, 3.chromosome sonucunda random sayı 4 e düştü, 4.chromosome sonucunda random -1 e düştü burada durduk)

Negatife düştüğünde o chromosome seçilir ve maxRandomNumber (İlk başta 100 olandan) dan o chromosome un çarktaki yüzdelik oranı çıkarılır (100 – 5 = 95). 2. Chromosome un seçilmesi için 95 max olacak şekilde random sayı üretilir ve üstteki işlem gerçekleşir. Bu sayede 2 tane chromosome seçilmiş olundu.

Çaprazlama: Seçilimden gelen 2 tane chromosome un gen sayısının yarısından sonraları yer değiştirilir. (16 gen varsa 1.chromosome un son 8 geni ile 2.chromosome un son 8 geni yer değiştirir)

1. abcdefgh – 2. qweqweqw ->1. abcdweqw – 2. qweqefgh)

Mutasyon: Mutasyon için öncelikle sayede two opt mutation yöntemini tüm chromosome lara uğratacak şekilde oluşturmuştum fakat belirli nesil tekrarından sonra belirli chromosome a doğru evrimleşmeye başladığı için normal mutation da ekledim bu sayede problem çözüldü. Generationdaki en iyi fitness değerine sahip chromosome u two opt mutation a, geri kalan chromosome ları da normal mutation a yolladım.

Two Opt Mutation: Chromosome un genleri aralarında sırayla yer değiştirir ve daha iyi bir fitness function a sahip bir dizilim bulunursa sabit kalır, bulunmazsa değişiklik geriye alınır. Tüm genler dolaşılır ve chromosome mevcut genleriyle en iyi dizilime sahip olacak şekilde dizilmiş olur.

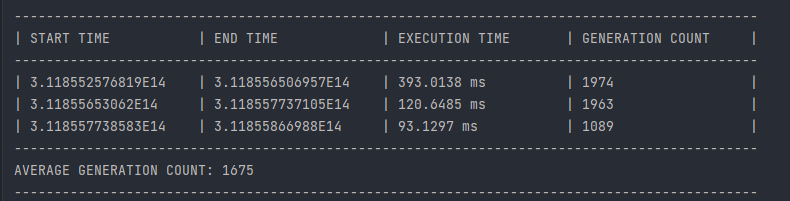
(Gchtewr->ehctGwr)

Normal Mutation: Constant değerler içerisinde mutation olasığı bulunuyor. Max 100 olacak şekilde random sayı oluşturuluyor ve random sayı olasılıktan küçükse chromosome normal mutation a uğruyor değilse uğramıyor. Normal mutationu chromosome un genlerinden 1 i rastgele seçilir ve asci tablosundan rastgele bir değerle değiştirilir. Bu sayede generation içindeki çeşitlilik sağlanmış olunur.

## 3.c Çözüm Süreleri Karşılaştırması

Gen sayısındaki yani şifredeki uzunluğun düşüşü şifrenin kaç generation da bulunduğu sayısında düşüş sağlar. Şifrenin uzunluğu azaldığı için daha az kombinasyonda sonuca ulaşabiliyoruz.

20 Chromosome 17 harfli çözüm



20 Chromosome 7 harfli çözüm

metin, elektronik donanım içeren bir resim

Açıklama otomatik olarak oluşturuldu

# 4) Makine Öğrenmesi

## 4.a Standardization ve Normalization Farkı ve Python Örneği

## 4.b Veri Setinin ve Problemin Kısa Anlatımı

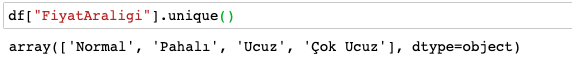
Veri seti 2000 tabletin teknik özelliklerini ve fiyat aralığını belirten bir tabloya ait. Verilen veri setine, göre verilen tablet özelliklerini sınıflandırılmamız bekleniyor.

Veri seti;

20 Öznitelik ve 2000 gözlemden oluşuyor.

Table

Description automatically generated

Özniteliklerden 2 tanesi boş değerler içeriyor. Özniteliklerden FiyatAraligi her bir gözlem için sınıflarını belirtiyor. Bu sınıflar şu şekilde;

Bu veri seti sınıflandırıcılara verilmeden önce ön işleme ve eksik verilerinin tamamlanması aşamalarından geçmiştir. Bu sebeple kategorik veriler sayısal değerlere dönüştürülmüştür.

## 4.c İki Farklı Sınıflandırıcı için Python Kodu

A picture containing graphical user interface

Description automatically generated

Graphical user interface, text, table

Description automatically generated with medium confidence

GridSearch kullanarak Desicion Tree için en iyi parametreleri araştırdık. Bunun sonucunda modeli tekrar eğittikGraphical user interface, text, application

Description automatically generated

Fakat bir değişim gözlemlemedik.

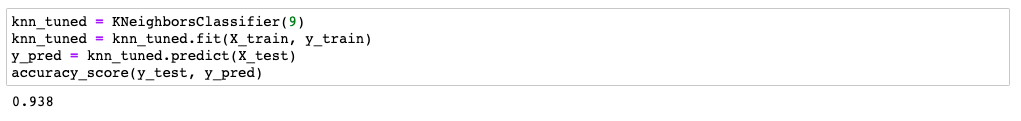
KNN

Graphical user interface, text, application, email

Description automatically generated

KNN içinde en iyi parametreleri araştırdık. Modelin hangi parametrelerde ne kadar iyi olduğunu gözlemlemek için grafik çizdirdik.Chart, line chart

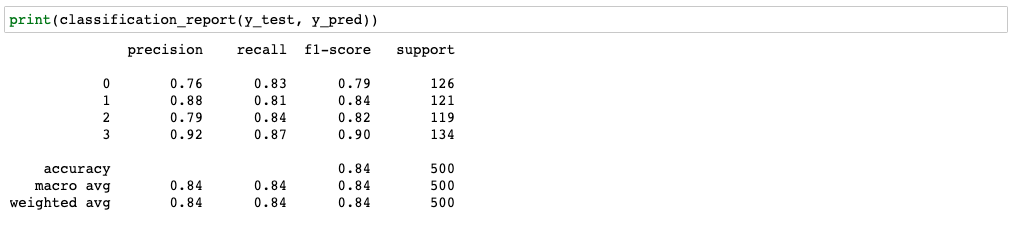
Description automatically generated

Grafikten yola çıkarak modeli tekrar eğittik.

## 4.d İki Farklı Sınıflandırıcı için Sonuçlar: Hata Matrisleri, Tablo

**Desicion Tree**

Graphical user interface, application

Description automatically generated

**KNN**

Graphical user interface, text, application

Description automatically generatedTable

Description automatically generated

## 4.e Kullanıcı tarafından verilen örneğin sınıflandırma ekran görüntüsü (konsol çıktıları)

**Desicion Tree**

Graphical user interface, text, application, email

Description automatically generated

0 -> Normal fiyat aralığı

**KNN**Graphical user interface, text, application, email

Description automatically generated

1 -> Pahalı fiyat aralığı

## 4.f ChatGPT Kullanımı ve Yorumlama

# 5) Öz değerlendirme Tablosu

**Açıklama kısmında yapıldı, yapılmadı bilgisi ve hangi maddelerin nasıl yapıldığı veya neden yapılamadığı kısaca yazılmalıdır.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **İstenen Özellik** | **Var** | **Açıklama** | **Tahmini Not** |
| **1a** | Algoritmalar + Karmaşıklıklar (10) |  |  |  |
| **1b** | Tanım ve Karşılaştırmalar (10) |  |  |  |
| **1c** | Araştırma ve Yorum (10) |  |  |  |
| **2** | Problem Çözme ve Kodlama (10) | ü | YAPILDI | 10 |
| **3** | Genetik Algoritmalar ile Şifre Kırma (15) | ü | YAPILDI | 15 |
| **4** | Makine Öğrenmesi (25) |  |  |  |
|  | Rapor (20) | ü | YAPILDI | 20 |
| **100 üzerinden Toplam Not:** | | | |  |