İçindekiler

**[YAPAY ZEKA](#_Toc377922103)** [2](#_Toc377922103)

[**SORU-1: Jensen Shannon Mesafesi (Jensen-Shannon Divergence) hakkında bilgi veriniz.** 2](#_Toc377922104)

[**SORU-2: Kullback Leibler Uzaklığı (kullback-leibler divergence) hakkında bilgi veriniz.** 3](#_Toc377922105)

[**SORU-3: Turing Testi hakkında bilgi veriniz.** 4](#_Toc377922106)

[**SORU-4: Sınıflandırma (Classification) hakkında bilgi veriniz.** 5](#_Toc377922107)

[**SORU-5: K Fold Cross Validation (K Katlamalı Çarpraz Doğrulama) hakkında bilgi veriniz.** 7](#_Toc377922108)

[**SORU-6: Naive Bayes ile Metin Sınıflandırılması hakkında bilgi veriniz.** 10](#_Toc377922109)

[**SORU-7: Naif Bayes Sınıflandırıcısı (Naive Bayes) hakkında bilgi veriniz.** 13](#_Toc377922110)

[**SORU-8: C4.5 Ağacı (C4.5 Tree) hakkında bilgi veriniz.** 16](#_Toc377922111)

[**SORU-9: Graph ve RDF Veritabanları hakkında bilgi veriniz.** 22](#_Toc377922112)

[**SORU-10: Karar Ağacı Öğrenmesi (decision tree learning) hakkında bilgi veriniz.** 25](#_Toc377922113)

[**SORU-11: Öbekağacı (dendrogram) hakkında bilgi veriniz.** 28](#_Toc377922114)

[**SORU-12: İleri ve geri zincirleme (Forward and Backward Chaining) hakkında bilgi veriniz.** 29](#_Toc377922115)

[**SORU-13: PROLOG hakkında bilgi veriniz.** 41](#_Toc377922116)

[**SORU-14: SimHash (Benzerlik Özeti) hakkında bilgi veriniz.** 48](#_Toc377922117)

[**SORU-15: Sezgisel Fonksiyonlar (Heuristic Functions) hakkında bilgi veriniz.** 50](#_Toc377922118)

[**SORU-16: Yinelemeli Derinlik Araması (Iterative Deepining Search) hakkında bilgi veriniz.** 55](#_Toc377922119)

[**SORU-17: Labirentte yol bulma kodu** 57](#_Toc377922120)

[**SORU-18: Işın Araması (Beam Search) hakkında bilgi veriniz.** 64](#_Toc377922121)

[**SORU-19: WEKA ile SVM hakkında bilgi veriniz.** 65](#_Toc377922122)

# **YAPAY ZEKA**

## **SORU-1: Jensen Shannon Mesafesi (Jensen-Shannon Divergence) hakkında bilgi veriniz.**

İki olasılık dağılımı arasındaki mesafeyi ölçmek için kullanılan yöntemlerden birisidir. Literatürde “bilgi çapı” (information radius , kısaca iRad ) veya ortalamaya olan toplam uzaklık (total divergence to average) olarak da geçmektedir.

Tanımı itibariyle Kullback Leibler Uzaklığının (kullback-leibler divergence) simetrik hali olarak düşünülebilir. Yani kullback leibler mesafesinde iki olasılık dağılımı olan P ve Q için P’den Q’ya olan mesafe (DKL(P||Q)) ile Q’dan P’ye olan mesafe (DKL(Q||P)) birbirine eşit olmamaktadır. Oysaki Jensen Shannon mesafesi bu sorunu çözerek simetrik bir mesafe fonksiynu tanımı yapar.

Temel tanımı aşağıdaki şekildedir:

Sigma cebirinde tanımlı bir A değişkeni için M_+^1(A)olasılık dağılımını ele alalım bu dağılım üzerindeki Jensen Shannon Divergence (JSD) değeri aşağıdaki şekilde hesaplanabilir:

JSD(P \parallel Q)= \frac{1}{2}D(P \parallel M)+\frac{1}{2}D(Q \parallel M)

Yukarıdaki gösterimde P ve Q şeklinde verilen iki olasılık dağılım fonksiyonu üzerinde M aşağıdaki şekilde tanımlıdır:

M=\frac{1}{2}(P+Q)

Yani M değeri aslında ortalama değerdir P ve Q’nun M değerine olan mesafesi üzerinde JSD fonksiyonu tanımlıdır.

Şayet M değeri, sayılabilir (countable) ise aşağıdaki şekilde daha özel tanımlanması mümkündür:

JSD(P_1, P_2, \ldots, P_n) = H\left(\sum_{i=1}^n pi_i P_i\right) - \sum_{i=1}^n pi_i H(P_i)

Yukarıdaki tanımda geçen H(P) değeri shannon entropisidir.

## **SORU-2: Kullback Leibler Uzaklığı (kullback-leibler divergence) hakkında bilgi veriniz.**

Olasılık teorisi ve bilgi teorisinde (information theory) geçen ve iki olasılık dağılımı arasındaki ilişkiyi (uzaklığı) formüllendiren yöntemin ismidir. Tanımı itibariyle, bilgi kazanımı (information gain) veya entropi kavramlarına yakındır ancak formüllendirmesi uzaklık (mesafe) değerinden sonuç döndürür.  
Literatürde mesafe anlamına gelen İngilizce “distance” kelimesinin ilk harfi olan D ile gösterilir ve DKL şeklinde ifade edilir. İndisteki KL harfleri Kullback Leibler isimlerinin baş harflerinden oluşmaktadır.

İki olasılık dağılım (probability distribution) olan P ve Q dağılımlarını inceliyor olalım.

DKL(P||Q) gösterimi ile P dağılımı kullanılarak Q dağılımının tahmin edilmesi halindeki bilgi kaybı (information loss) ifade edilir.

Burada önemli bir nokta, Kullback Leibler uzaklığının (Kullback-Leibler Divergence), her ne kadar ismi uzaklık (veya mesafe) olarak geçiyor olsa da aslında bu mesafenin, bizim alıştığımız mesafe kavramından biraz farklı olması ve mesafenini simetrik olmamasıdır. Örneğin DKL(P||Q) mesafesi ile DKL(Q||P) mesafesi eşit olmak zorunda değildir (iki dağılım aynı iken eşit olabilirler ama farklı dağılımlar için iki değer birbirinden farklıdır).

Olasılık dağılımları kesikli (ayrık, discrete) veya sürekli (continous) olabilir. Buna göre ayrık dağılımlar için aşağıdaki yöntemle hesaplama yapılabilir:

D_{KL}(P|Q) = sum_i lnleft(frac{P(i)}{Q(i)}right) P(i).!

Kısaca, iki dağılımın birbirine her noktadaki oranları alınıp bu oranın logaritmalarının toplamına eşittir.

Genellikle bilgisayar bilimlerinde ayrık sistemler üzerinde çalışıldığından bu yazıda devamlı (continous) sistemler hakkında bilgi verilmeyecektir ancak çok kaba bir şekilde, ayrık sistemlerdeki toplam sembolünün, devamlı sistemlerde integral olduğunu söyleyebiliriz.

Kullback-leibler mesafesi her zaman için eksi olmayan bir değer (sıfır veya daha büyük) döndürür.

D_{KL}(P|Q) geq 0, ,

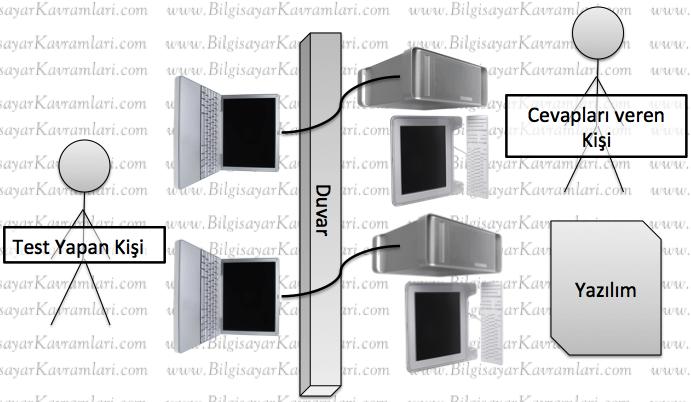
Mesafenin sıfır olması, ancak iki dağılımın birbirinin aynısı olması durumunda mümkündür. Bunun dışındaki durumlar için sonuç pozitif bir reel sayı çıkacaktır.

## **SORU-3: Turing Testi hakkında bilgi veriniz.**

Bu yazının amacı, yapay zeka (artificial intelligence) konusunun temellerini oluşturan bir test kavramından bahsetmektir. Allen Turing tarafından ortaya atılan bir teste göre, bir bilgisayarın, bir insan gibi davranabilmesinin tanımını yapılmıştır.

Yani, yapay zeka çalışmalarının hedeflerinden birisi, bir gün insan gibi çalışan bir bilgisayar yapmak olarak görülürse, bu durumda bir bilgisayarın, insan gibi çalışabilmesi nasıl tanımlanabilir?

Turing, bu durumu basit bir testle açıklar. Bir duvarın arkasında iki bilgisayar ve bilgisayarlardan birsinin klavyesinde yazı yazan bir kişi, diğerinde ise çalışan bir yazılım olması durumunda, verilen sorulara bakarak bir kişinin bu bilgisayarlardan hangisinde insanın cevap verdiğini, hangisinde bir yazılımın cevap verdiğini anlayıp anlayamamasıdır.



Şekil 1: Turing testi

Yukarıdaki şekilde gösterildiği üzere, duvarın sol tarafında, testi yapan bir kişi bulunmaktadır. Bu kişi önündeki iki bilgisayardan birncisi duvarın arkasında, gerçek bir insanın cevap yazdığı bir bilgisayar, diğeri ise duvarın arkasında bir yazılımın cevap verdiği bir bilgisayar bağlıdır. Buna göre test yapan kişi, yazdığı ve aldığı sorulardan acaba hangi bilgisayarın arkasında gerçek bir insan, hangisinin arkasında bir yazılım olduğunu bulabilecek midir?

Burada sorulabilecek sorularda bir sınır bulunmamaktadır. İstenen sorular sorulabilir. Örneğin 4096 sayısının karekökünü bir bilgisayar insana göre çok daha hızlı verecektir. Veya “bugün ne var ne yok” gibi bir soruyu bir yazılımın anlaması daha uzun zaman alacak (veya hiçbir zaman mümkün olmayacaktır).

Bu testin ortaya konmasının ardından, bu testin hiçbir zaman geçilemeyeceğini düşünenler gibi bir gün bu testi geçen bir yazılımın olacağını düşünenler de olmuştur.

## **SORU-4: Sınıflandırma (Classification) hakkında bilgi veriniz.**

Bu yazının amacı, bilgisayar bilimleri ve iş zekası (business intelligence) gibi disiplinlerin ortak çalışma alanlarından olan veri madenciliği (data mining) konusunda kullanılan metotlardan birisi olan sınıflandırma (classification) kavramını açıklamaktır.

Sınıflandırma kavramı, basitçe bir veri kümesi (data set) üzerinde tanımlı olan çeşitli sınıflar arasında veriyi dağıtmaktır. Sınıflandırma algoritmaları, verilen eğitim kümesinden bu dağılım şeklini öğrenirler ve daha sonra sınıfının belirli olmadığı test verileri geldiğinde doğru şekilde sınıflandırmaya çalışırlar.

Veri kümesi üzerinde verilen bu sınıfları belirten değerlere etiket (label) ismi verilir ve gerek eğitim gerekse test sırasında verinin sınıfının belirlenmesi için kullanılırlar.

Konunun daha kolay anlaşılabilmesi için bir örnek üzerinden anlatmaya çalışalım.

Örneğin aşağıdaki şekilde öğrencilerden toplanan bir veri kümemiz bulunsun.

|  |  |  |  |
| --- | --- | --- | --- |
| Yaş | Boy | Kilo | Cinsiyet |
| 20 | 175 | 70 | Erkek |
| 21 | 179 | 80 | Erkek |
| 19 | 162 | 50 | Kız |
| 22 | 169 | 55 | Kız |
| 20 | 183 | 90 | Erkek |
| 19 | 181 | 75 | Erkek |
| 21 | 171 | 57 | Kız |

Örneğin problemimizi şu şekilde tanımlayalım. Bir sınıflandırıcı yöntemimiz, yukarıdaki kümeye bakarak verilen yaş boy ve kilo değerlerine göre bir öğrencinin cinsiyetini öğrenecek olsun. Yani yukarıdaki veri kümesini bir eğitim kümesi olarak kullanacağız. Ardından gelen yeni bir kayıt için, yaş, boy ve kilo değerleri verildiğinde, sınıflandırıcımız cinsiyetini otomatik olarak tahmin edecek (prediction).

Çok sayıdaki sınıflandırma algoritmalarından basit birini seçelim. Diyelim ki sınıflandırma algoritmamız, verilen etiketteki değerlerin ortalamasını alacak ve bu ortalama değer, öğrendiği değer olacak. Ardından gelen test değerleri için bulmuş olduğu ortalamaya uzaklığına bakacak ve kime yakınsa o etiketten kabul edecek.

Yukarıdaki veri kümesini iki sınıf için ikiye bölelim ve ortalama değerlerini alalım:

Erkekler için öğrenme işlemimiz aşağıda verilmiştir:

|  |  |  |  |
| --- | --- | --- | --- |
| Yaş | Boy | Kilo | Cinsiyet |
| 20 | 175 | 70 | Erkek |
| 21 | 179 | 80 | Erkek |
| 20 | 183 | 90 | Erkek |
| 19 | 181 | 75 | Erkek |
| 20 | 179,5 | 78,75 | Ortalama |

Aynı işlemin kızlar için olanı aşağıdaki şekildedir:

|  |  |  |  |
| --- | --- | --- | --- |
| Yaş | Boy | Kilo | Cinsiyet |
| 21 | 171 | 57 | Kız |
| 19 | 162 | 50 | Kız |
| 22 | 169 | 55 | Kız |
| 20,66666667 | 167,3333333 | 54 | Ortalama |

Sonuç olarak algoritmamız erkekler için (20,179.5, 78.5) değerlerini öğrenirken, kızlar için (20.66, 167.33, 54) değerlerini öğreniyor. Diyelim ki yeni gelen ve test edilmesini istediğimiz değer de, yaş : 21, boy: 165, kilo 60 değerlerinde bir kişi olsun. Şimdi algoritmamız öğrendiği değerlere göre bu yeni gelen kişinin cinsiyetini tahmin etmeye çalışacak. Basitçe her değere olan mesafeyi hesaplayacak (burada da çok farklı mesafe hesaplama algoritmaları olmasına karşılık biz yine amacımız temel kavramlar olduğu için konuyu basit tutarak gauss mesafesini (gaussian distance) kullanalım).

Algoritmamızın Erkek tanımından öğrendiği değerler ile yeni gelen kişinin mesafesini hesaplayalım:

http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1364716991_0__.gif

Benzer şekilde kızlar için öğrendiğimiz değere olan mesafesini hesaplamaya çalışalım.

http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1364716991_1__.gif

Buna göre algoritmamızın verdiği değer, erkeklere olan mesafesinin 23.72 olduğu ve kızlara olan mesafesinin 6.44 olduğudur. Demek ki algoritmamız yeni gelen kişiyi kız sınıfından tanımlamıştır.

İşte sınıflandırma algoritmalarının çalışması, yukarıda da anlatıldığı gibi en basit anlamıyla, 2 aşamadan oluşur.

* Eğitim verisi üzerinden öğrenme
* Öğrenilen değerlerle test verisi üzerinde sınıflandırma

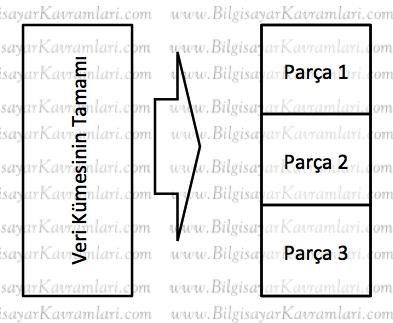
Ancak veri madenciliği ve iş zekası çalışmalarında sınıflandırma sadece çalışmanın bir türü ve ufak bir parçası olmaktadır.

## **SORU-5: K Fold Cross Validation (K Katlamalı Çarpraz Doğrulama) hakkında bilgi veriniz.**

Bu yazının amacı, literatürde k-fold cross validation (k katlamalı çarpraz doğrulama) yöntemi olarak geçen yöntemi anlatmaktır. Veri madenciliği çalışmalarında, uygulanan yöntemin başarınının sınanması için, veri kümesini eğitim ve test kümeleri olarak ayırılmaktadır. Bu ayırma işlemi çeşitli şekillerde yapılabilir. Örneğin veri kümesinin %66′lık bir kısımını eğitim %33′lük bir kısmını ise test için ayırmak ve eğitim kümesi ile sistem eğitildikten sonra test kümesi ile başarısının sınanması kullanılabilecek yöntemlerden birsidir. Bu eğitim ve test kümelerinin rastgele olarak atanması da farklı bir yöntemdir. Fakat bu yazının konusu eğitim ve test kümelerinin nasıl katlanarak değiştirildiğidir.

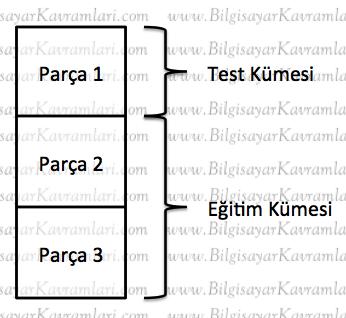
K-katlamalı çarpraz doğrulama yönteminde öncelikle bir k değeri seçilir. Ben yazıda anlatması kolay olsun diye 3 olarak seçeceğim. Ancak belirtmem gerekir ki literatürde en çok tercih edilen k değeri 10′dur.

K= 3 için veri kümemizi öncelikle 3 eşit parçaya bölüyoruz. Diyelim ki veri kümemizde 300 kayıt bulunuyor olsun. Eşit olarak 100′er parçalık bölümlere parçalıyoruz.



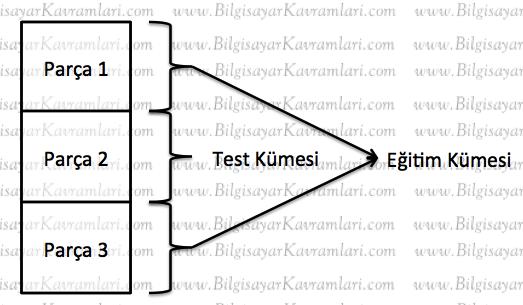
Şekil 1: Veri kümesi

Yukarıdaki şekilde gösterildiği üzere veri kümesi verilen k değeri kadar parçaya ayrıldıktan sonra k-katlamalı çarpraz doğrulama sistemi çalışmaya başlar. Yukarıdaki örnek için 3-katlamalı çarpraz doğrulama yöntemi denilebilir. Öncelikle parçalardan birisi test için seçilip geri kalanı eğitim için kullanılacaktır. Burada hangi parçadan başlandığının bir önemi yoktur. Diyelim ki ilk parçayı test için seçip ikinci ve üçüncü parçaları eğitim için kullanmaya karar veriyoruz.



Şekil 2: Veri kümesi

Yukarıdaki seçimin ardından, sistemimizi çalıştırıyoruz. Örneğin amacımız sınıflandırma yapmaksa bir sınıflandırma algoritmasını çalıştırıyor ve bir sonuç elde ediyoruz. Buna kısaca birinci sonuç anlamında S1 diyelim. Ardından aynı işlem ikinci parça seçilerek tekrarlanıyor.



Şekil 3: Kümeler

Yine sınıflanıdrıcımızı çalıştırıyor ve bir netice elde ediyoruz. Diyelim ki bu neticenin değeri de S2 olsun. Ardından son çalıştırma için geriye kalan 3. parçayı test kümesi olarak ele alıyoruz:



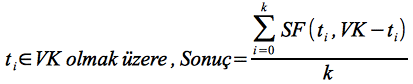
Şekil 4: Kümeler

Son olarak bu seçimden elde edilen değere de S3 diyelim.

Neticede 3 kere aynı yöntemi, 3 farklı eğitim ve test kümelerinde çalıştırmış oluyoruz. Sistemin genel başarısı veya genel hatası (error rate) ise, elde edilen 3 sonucun ortalaması olarak hesaplanıyor. İşte bu son adımda ortalama alındığı için ve ortalama hesaplanırken kullanılan toplama işleminin yer değiştirme özelliği olduğu için, hangi parçadan başlandığının aslında bir önemi yoktur.

Özetleyecek olursak. K katlamalı çarpraz doğrulama için k-kere yöntem çalıştırılır. Her adımda veri kümesinin 1/k kadar, daha önce test için kullanılmamış parçası, test için kullanılırken, geri kalan kısmı eğitim için kullanılır.

Buna göre her aslında yapılan işlem aşağıdaki şekilde daha resmi bir gösterimle yazılabilir:



Şekil 5: Formül

Buradaki S *F(test, eğitim)*, sınıflandırma fonksiyonu , *VK*, veri kümesi, *k* , kaç parça katlama kullanıldığı ve *t* ise veri kümesi üzerinden seçilen her bir test kümesi olarak verilmiştir. Yukarıda formülize edildiği üzere, sonuç bütün sınıflandırma fonksiyonlarının performanslarının toplamının, k sayısına bölünerek ortalaması alınmasıdır.

## **SORU-6: Naive Bayes ile Metin Sınıflandırılması hakkında bilgi veriniz.**

Bu yazının amacı, naif bayes sınıflandırıcısının (naive bayes) metinler üzerinde nasıl kullanıldığını açıklamaktır. Oldukça basit ve etkili bir metin madenciliği yöntemi olan naif bayes sınıflandırıcısını anlamak için bir örnek kullanalım.

Örneğin iki metin aşağıdaki şekilde verilmiş olsun:

metin 1 : java bazı bilgisayar mühendisliği bölümlerinde eğitimi verilen bir programlama dilidir.

metin 2 : bazı bilgisayar mühendisliği projelerinde java programlama dili kullanılmaktadır.

Bu metinlerin, kavram metin masfufu (term document matrix) daha önceki bir yazıda gösterilmiştir. Şimdi amacımız yeni gelen aşağıdaki metnin, hangisine daha yakın olduğunu bulmak olsun.

Metin 3 : bilgisayar kavramları eğitimi

Problemi çözmeye başlamadan önce Naive Bayes yönteminin klasik formülünü hatırlayalım:

http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1360341687_0__.gif

Şekil 1: Naive Bayes formülü

Bu klasik formülde bir koşullu olasılığın, nasıl küme olasılıklarına indirgendiği gösterilmektedir.

Metin madenciliğinde (text mining) önemli konulardan birisi de metin içerisinden özelliklerin nasıl çıkarılacağıdır. Bu yazı kapsamında, konuyu dağıtmamak için en basit yöntem olarak kelime seviyesinde çalışacağız. Yani bir kelimenin bir sınıfı belirlemede kullanılmasını hedefliyoruz.

Yine olayı basitleştirmek için yukarıdaki bayes formülünde olduğu üzere 2 sınıf var olduğunu kabul edeceğiz.

Bu durumda herhangi bir kelimenin herhangi bir sınıfta bulunma olsılığı aşağıdaki şekilde gösterilebilir:

http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1360341687_1__.gif

Yani diğer bir deyişle S sınıfı gerçekleşen olaysa, ki kelimesinin bu olayın içinde yer alma olsılığıdır.

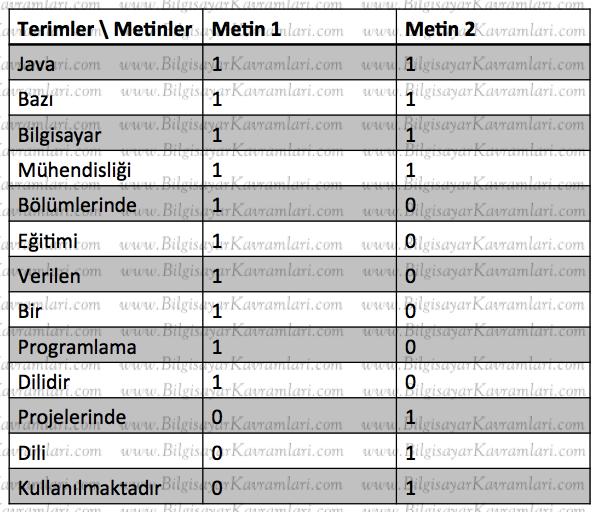
Bu durumda naive bayes sınıflandırma yöntemini kullanırsak (lütfen ilgili yazı için tıklayın) kelime bazlı sınıflandırıcımızın aşağıdaki şekilde olduğunu söyleyebiliriz.

http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1360341687_2__.gif

Yani diğer bir deyişle, sistemimizde sınıflandırılmak istenen X metninin S sınıfında olma olasılığı, bu metindeki bütün kelimelerin S sınıfında olma olasılıklarının çarpımıdır.

Neticede elimizdeki bütün sınıflar için metin denenerek en yüksek olasılığa sahip sınıfa aittir denilebilir.

Örneğimize dönecek olursak metinlerdeki kelimeleri listeleyelim:



Şekil 2: Metinlerdeki kelimeler

Şimdi bu listeye göre metin1′de 9 kelime bulunuyor ve hepsinin frekansı (sıklığı) aynı, dolayısıyla bütün kelimeler için

http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1360341687_4__.gif

oranındadır denilebilir. Aynı durum Metin2 için de geçerlidir ve 7 kelimenin tamamı için:

http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1360341687_5__.gif

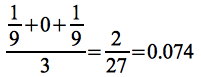
değerindedir denilebilir. Neticede yeni gelen ve sınıflandırılmak istenen Metin 3 için bütün kelimelerin değerlerine bakacağız.

M3 = {bilgisayar, kavramları, eğitimi}

http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1360341687_6__.gif, http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1360341687_7__.gif, http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1360341687_8__.gif

Burada sık karşılaşılan bir problem olan 0 durumunu düzeltmek için iki yöntemden birisi kullanılır, ya normalleştirme yapılarak bu değer sistemden çıkarılır, ya da çok düşük bir değer konulur. Biz normalleştirme yapalım:

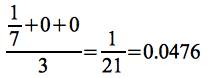
http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1360341687_9__.gifşeklinde ortalama alıyoruz.

Olarak bulunur.

Şimdi gelelim ikinci metin için aynı hesabı yapmaya:

http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1360341687_11__.gif, http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1360341687_12__.gif, http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1360341687_13__.gif

Bu durumda normalleştirilmiş değer:

olarak bulunur. Demek ki yeni gelen metin 3, metin 1′e daha yakındır denilebilir. Sınıflandırmamız neticesinde bir sınıf seçilmesi istenseydi, metin 1′in bulunduğu sınıfta olduğu söylenebilirdi.

## **SORU-7: Naif Bayes Sınıflandırıcısı (Naive Bayes) hakkında bilgi veriniz.**

Bu yazının amacı, literatürde naif Bayes (Naive Bayes) olarak bilinen algoritmanın, sınıflandırma (classification) için kullanılma şeklini açıklamaktır. Herhangi bir sınıflandırma probleminde olduğu gibi, amacımız birden fazla özelliği taşıyan bir yöney (vektör) kullanarak verilen bilgilerden bir eğitim oluşturmak ve bu eğitim neticesinde gelen yeni verileri doğru bir şekilde sınıflandırmaktır.

Sınıflandırma işlemi sırasında, nasıl bir yol izlendiğini bir örnek üzerinden açıklamaya çalışalım.

Örneğimiz, aşağıda verilen tablodaki verilerden bir sınıflandırıcı eğitimi olsun (train).

|  |  |  |  |
| --- | --- | --- | --- |
| Kısım | Maaş | Yaş | İş Tecrübesi |
| Yazılım | 3000 | 26 | 4 |
| Muhasebe | 1500 | 22 | 2 |
| Yazılım | 5000 | 30 | 9 |
| Muhasebe | 2000 | 30 | 7 |
| Muhasebe | 500 | 18 | 3 |
| Yazılım | 2000 | 20 | 2 |
| Yazılım | 7000 | 29 | 5 |
| Muhasebe | 6000 | 45 | 15 |

Yukarıda, sadece muhasebe ve yazılım kısımlarında çalışan kişilerin maaş, yaş ve iş tecrübelerini içeren temsili bir tablo verilmiştir. Buna göre aşağıdaki şekilde bize bir bilgi verilse:

Maaş : 3000, Yaş : 30, Tecrübe : 5yıl

Bu kişinin hangi kısımda çalıştığını acaba bulabilir miyiz?

Öncelikle eğitim ile işe başlayalım sonra da testimizi yaparız. Basitçe veri kümemizideki (data set) Yazılım ve Muhasebe kısımlarının ortalama ve varyans değerlerini hesaplıyoruz. Bu değerler aşağıdaki şekildedir:

|  |  |  |  |
| --- | --- | --- | --- |
| Muhasebe | 2500 | 28,75 | 6,75 |
| Yazılım | 4250 | 26,25 | 5 |
| Muhasebe | 5833333,333 | 142,25 | 34,91666667 |
| Yazılım | 4916666,667 | 20,25 | 8,666666667 |

Yukarıdaki ilk iki satır ortalama ve ikinci iki satır ise varyans değerleridir. Bu değerleri basitçe Excel ile hesapladık.

Şimdi beklenen değeri hesaplayacağız. Yani gelen test verimizin Muhasebe kısmında birisine ait olması veya Yazılım kısmından birisine ait olması için beklenen durum hesabı yapacağız.

Hesaplamaya geçmeden önce yazının konusu olan naive bayes kavramını hızlıca açıklayalım. Aslında naive bayes sınıflandırıcısı basiçte bütün koşullu olasılıkların çarpımıdır.

Aşağıdaki şekilde gösterilebilir:

http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1360323659_0__.gif

Bu formülde görüleceği üzere s1′den sn’e kadar olan sınıflar arasından bir seçim yapılırken aslında bu sınıfın olasılık değeri ve bu sınıfları yerine getiren k koşulları için çarpımından bir farkı yoktur.

Yani diğer bir deyişle her sınıfın bir koşullu olasılık değeri vardır ve biz sınıflardan hangisine ait olduğunu bulmak için bu koşullu olasılık değerlerini çarparız.

Bu durumda bizim formülümüz aşağıdaki şekildedir denilebilir:

http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1360323659_1__.gif

Yani bir kişinin yazılım kısmında olduğunu anlamak için öncelikle yazılım kısmında olan kişilerin oranını buluyoruz, P(Yazılım), ardından yazılım kısmındaki kişiler için verilen maaş, yaş ve iş tecrübesine göre koşullu olasılıklarını bulup bunu normalleştirme değerine bölüyoruz.

Benzer şekilde muhasebe kısmındaki kişilere ait beklenti de aşağıdaki gibi hesaplanabilir

http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1360323659_2__.gif

Buradaki fark, verilen bilgilerin muhasebe kısmındaki kişilere göre koşullu olasılığının alınmasıdır.

Normalleştirme değeri ise sistemimizde bulunan bütün ihtimalleri içeren ve beklenti değerlerini normalleştiren değerdir. Aşağıdaki şekilde hesaplanabilir:

normalleştirme = P(Muhasebe) p(maaş | Muhasebe) p (Yaş | Muhasebe) p (iş tecrübesi | Muhasebe) + {P(Yazılım) p(maaş | yazılım) p (Yaş | yazılım) p (iş tecrübesi | yazılım)

Şimdi iki sınıfımız olduğu ve aslında ikisi arasında seçim yapacağımız için, iki sınıfı da bölen ve pozitif çıkmasını beklediğimiz normalleştirme değerini göz ardı edebiliriz.

Sırasıyla olasılıkları hesaplayalım:

P(Yazılım) = 8 kişiden 4′ü yazılım kısmında

P(Yazıılm) = 0.5

Benzer şekilde,

P(Muhasebe) = 0.5

olarak buluruz. Ardından koşullu olasılık değerlerini hesaplayacağız. Bu aşamada gauss dağılımını kullanmak isteyelim (farklı dağılımlar kullanılabilir ve başarı bu dağılıma göre değişebilir) ve dağılım fonksiyonunu hatırlayalım:

http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1360323659_3__.gif[varyans](http://www.bilgisayarkavramlari.com/2012/07/16/varyans-variance/), maaş ve ortalama değerlerini yazacak olursak:

http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1360323659_4__.gif

Yani sonuç olarak 0.0000000687′e yakın bir değer bulunuyor. Bunun anlamı, verilen 3000 lira maaş ile çalışan kişinin Gauss dağılımında (dağılımın toplam alanının 1 olduğunu hatırlayınız), yazılım kısmında çalışan birisi olması ihtimalinin 6.87E-7 olduğudur. Daha doğru bir ifadeyle, bir kişinin yazılımcı olduğunu kabul edersek (verilen koşul) bu kişinin 3000 lira maaş alma durumunu hesaplamış olduk. Şimdi aynı maaş değerine sahip kişinin muhsabe kısmında çalışma olasılığını hesaplayalım.

http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1360323659_5__.gif

Yukarıdaki ikinci hesaplamanın sonucuna bakarak aslında bu maaşın, yazılım kısmındaki kişilere daha uygun olduğunu söyleyebiliriz.

Benzer hesaplamaları diğer koşullu olasılık durumları için de yaparsak aşağıdaki gibi bir tablo elde edebiliriz:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Maaş | Yaş | Tecrübe |
| Muhasebe | 6,84074E-08 | 0,002805118 | 0,011414151 |
| Yazılım | 8,11614E-08 | 0,019705849 | 0,046043474 |

Sonuçta naive bayes yöntemine göre bu verilen olasılıkların çarpımlarını alacağız:

http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1360323659_6__.gif

olduğunu hatırlayalım. Bu durumda beklenti(yazılım) aşağıdaki şekilde yazılabilir:

http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1360323659_7__.gif

Burada normalleştirme değeri daha önce de belirtildiği üzere hesaba katılmamıştır. Benzer şekilde muhasebe kısmına ait beklenti de aşağıdaki şekilde hesaplanabilir.

http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1360323659_8__.gif

Görüldüğü üzere muhasebe beklentisi, yazılım beklentisinin yaklaşık 20 misli daha yüksek çıkmıştır. Demek ki naive bayes sınıflandırmasına göre bu kişinin muhasebe kısmında çalıştığını söyleyebiliriz, en azından beklentimiz bu yönde olur.

## **SORU-8: C4.5 Ağacı (C4.5 Tree) hakkında bilgi veriniz.**

Bu yazının amacı, karar ağaçlarına (decision tree) bir örnek olaran C4.5 ağacını açıklamaktır. C4.5 ağacı, ID3 ağacının geliştirilmiş bir hali olarak düşünülebilir ve daha önce bu konuda yayınlanan ID3 ağacı başlıklı yazıyı okumanızda yarar vardır. Bu yazıda iki ağacı karşılaştırarak konu anlatılacaktır.

C4.5 ağacının ID3 ağacından en büyük farkı normalleştirme (normalization) kullanıyor olmasıdır. Yani ID3 ağacı üzerinde e ntropi hesabı yapılır (veya bilgi kazanımı (information gain)) ve bu değere göre karar noktaları belirlenir. C4.5 ağacında ise entropi değerleri birer oran olarak tutulur. Ayrıca ağaç üzerinde erişim sıklıklarına göre alt ağaçların (subtree) farklı seviyelere taşınması da mümkündür. C4.5 ağacının diğer bir farkı ise tam bu noktada orataya çıkar ID3 ağacının yaklaşımından farklı olarak C4.5 ağacında budama (prunning) işlemi yapılmaktadır.

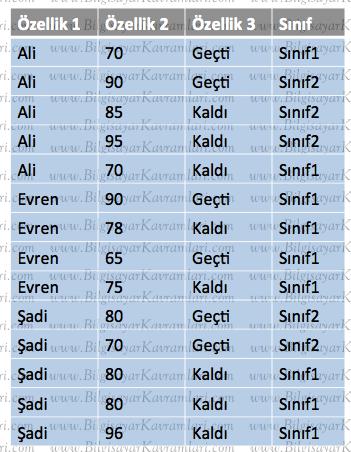


Şekil 1: İşlemler

Bu farklılıklardan bahsettikten sonra nasıl çalıştığını hızlıca açıklamaya çalışalım.

* Her adımda bütün özellikler kontrol edilir
* Her özelliğin normalize edilmiş bilgi kazanımı (information gain) hesaplanır
* En iyi bilgi kazanımını veren özellik karar ağacında karar olarak taşınır.
* Ardından bu yeni karar düğümünün altında bir alt liste oluşturularak alt karar ağacı inşa edilir.

Konuyu bir örnek üzerinden anlamaya çalışalım. Örnek olarak aşağıdaki veri kümesini (data set) ele alalım:



Şekil 2: Örnek veri kümesi

Yukarıdaki veri kümesi için bir C4.5 ağacı oluşturmak istiyor olalım. Yapacağımız ilk adım bilgi kazanımını (information gain) hesaplamaktır. Bu adımda bilgi kazanımının formülünü hatırlayalım. Bilgi kazanımı hesaplanırken, o anda veri kümesinde bulunan bütün veriler ve hesaplanması istenen belirli bir verinin üzerinden gidilir. Bu hesaplaması yapılacak olan belirli veriye örnekleme (misal, sampling) ismi verilir ve bütün veri kümesi üzerinden bu örneklemeye ait hesaplama yapılır.



Bilgi (information) hesaplaması sırasında kullanılacak olan formül yukarıdaki şekildedir. Buna göre herhangi bir misal (M ile gösterilmiştir) için o sınıftaki (S ile gösterilmiştir) değerlere göre frekansına bakılır. Ayrıca yukarıdaki formülde |M| değeri, o sınıftaki misallerin sayısını ifade etmektedir.

Yukarıdaki şekilde her örnek için bilgi değeri hesaplandıktan sonra kazanım (gain) hesaplanması mümkündür.

Genelde tam bu adımda bilgi parçalara bölünür ve bölünen parçalar (partition) üzerinden işlem yapılır. Bu durum için ise hesaplama aşağıdaki şekilde yapılabilir:



Yukarıdaki formülde her bir i parçası için yapılan bilgi hesaplaması verilmektedir.

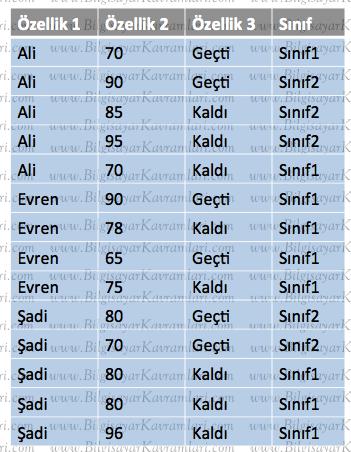
Kazanım ise bu durumda aşağıdaki şekilde hesaplanabilir:

http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1352757788_4__.jpg

Yani herhangi bir X özelliği için kazanım değeri, o özelliğin bağlı olduğu bütün parça ve sadece o özelliği ilgilendiren parça arasındaki farka eşittir. Bu iki değerin hesabı da yukarıda verilmiştir (yazıdaki ilk ve ikinci formüller).

Şimdi yukarıda anlattığımız bu değerlerin gerçek bir uygulama üzerinden nasıl hesaplandığını görelim.

Örnek veri kümemiz aşağıdaki şekilde olsun:



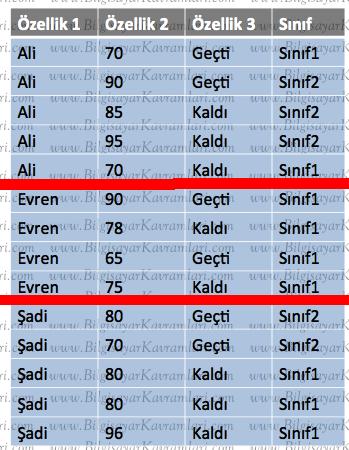
Şekil 3: Örnek veri kümesi

Örneğin sınıf değerinin bilgi kazanımını (information gain) hesaplamak istiyor olalım. Yukarıdaki formüle göre, 14 toplam satırdan 5 tanesi sınıf 2 ve 9 tanesinin sınıf 1 olduğunu dikkate alarak aşağıdaki eşitliği yazıyoruz. Önce bilgi değerlerini hesaplayacak sonra da kazanımı bulacağız:



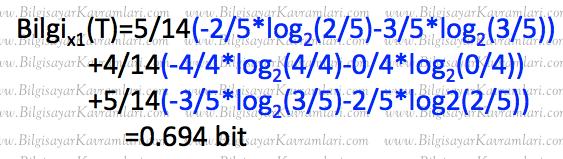
İlk bilgi değeri bütün parçanın hesaplandığı yani 14 satırın tamamının dikkate alındığı ve 9/14 ve 5/14 olarak iki ihtimalin hesaba katıldığı durumdur. Bu durum aynı zamanda entropi olarak da düşünülebilir.

İkinci bilgi hesabımızda özellik 1 kullanılacak. Buna göre veri kümemizin ilk 5 satırında Ali, sonraki 4 satırında Evren ve son 5 satırında Şadi özellikleri var. Buna göre tabloyu 3 parçaya bölersek :



Şekil 4: Bölünmüş tablo

Yukarıdaki yeni tabloya göre her özellik parçasının ayrı ayrı hesaplanarak denklemde yerine yazılması gerekir:



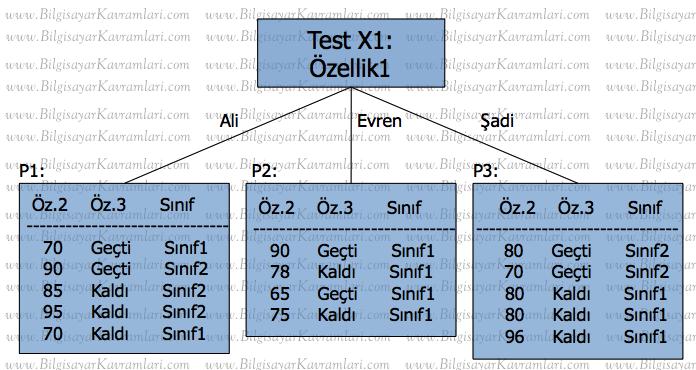
Yukarıdaki formülde mavir renkle belirtilen durum 1. özellik için (x1) 5, 4 ve 5 parçadan oluşan ve her parça için ayrı ayrı Sınıf1 ve Sınıf2 değerlerinin sayıldığı durumdur. Yani ilk 5 satırlık parçanın 2 satırı Sınıf1 ve 3 satırı Sınıf2 olduğu için 2/5 ve 3/5 şeklinde iki değer alınmıştır. Diğerleri de benzer şekilde hesaplanmıştır.

Son adımda bu iki değer arasındaki farkı hesaplayabiliriz:

http://www.bilgisayarkavramlari.com/wp-content/uploads/sbres_1352757788_9__.jpg

olarak bulunur.

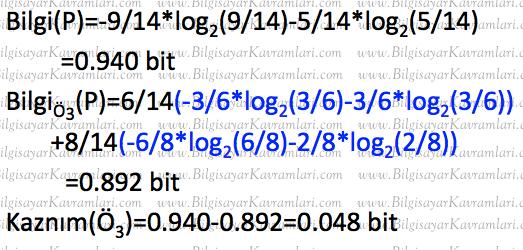
Yukarıda bulunan bilgi kazanımı, bütün veri kümesindeki Özellik1 için bütün sınıflar arasındaki kazanımı göstermektedir. Bu değerleri kullanarak aslında veri kümemizi 3 parçaya bölmüş ve her birisi için bilgi kazanımının olası değerini hesaplamış oluyoruz:



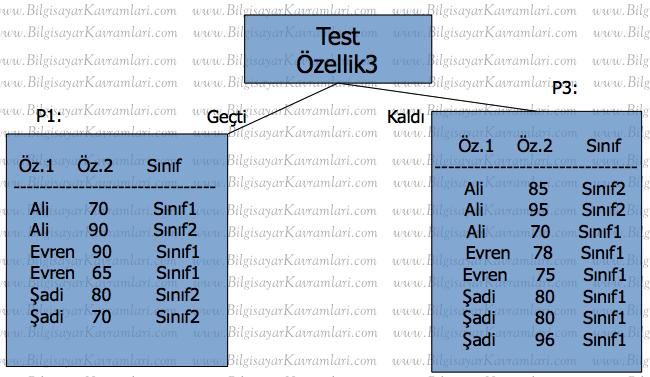
Şekil 5: Karar ağacının alabileceği olasılıklardan biri

Yukarıdaki gösterim karar ağacının alabileceği olasılıklardan birisidir ve ağacın özellik 1′de bulunan isimlere göre karar noktası oluşturulması halinde alacağı vaziyeti gösterir.

Şimdi aynı hesabı, Özellik 3 için yapalım. Burada geçti / kaldı ihtimalleri bulunuyor dolayısıyla ağaç 2 dala ayrılabilir. Ancak biz önce hesabımızı yapalım.



Özellik 3 için 3/6 ve 6/8 olmak üzere iki parça bulunuyor ve her ikisi için de hesap yapılıp toplam bilgiden çıkarıldığında kazanım değeri olarak 0.048 bit bulunuyor. Bu değeri yorumlamadan önce ağacın şu anda hesaplanan halini göstermeye çalışalım:



Şekil 6: Ağacın ilk karar noktası

Yukarıdaki şekilde ağacın ilk karar noktasını Geçti / Kaldı ihtimalleri üzerine kuracak olursak bilgi kazanımı olarak 0.048 beklenmektedir.

Bu durumda C4.5 ağacı en yüksek kazanıma sahip olan değeri alacaktır. Bu değer Özellik 1 için isimler olduğundan karar ağacının bu adımda Özellik 1′e göre karar noktası eklemesi yerinde olacaktır.

Ardından diğer adımlar için benzer şekilde hesaplamalar yapılarak ağacın karar noktaları oluşturulmaya devam edecektir.

C4.5 Ağacının önemli bir diğer özelliği ise budama işlemidir. Esas olarak ağaçlarda iki tip budama yapılabilir. Birisi ön budama (preprunning) diğeri ise son budama (post prunning). C4.5 ağacı son budama (postprunning) yöntemini tercih etmektedir.

Hemen burada karar ağaçlarında (decision trees) ön budama ve son budamanın nasıl yapıldığından bahsedelim. Ön budama, genelde ağaç oluşturulurken bazı dalların oluşturulmaması yönündedir. Örneğin bazı dallar anlamsız olacağından veya hiç eleman içermeyeceğinden oluşturulmaz. Son budama ise ağacın bütün dallarını oluşturur ve sonra bazı şartlara göre budama yapar. Burada da eleman içermeyen dallar budanabileceği gibi, bazı durumlarda istatistiksel yaklaşımlar da kullanılabilir. Örneğin yukarıdaki veri kümemizi Özellik 2′yi kullanarak 10luk dilimlere bölmek isteyelim. 100-90 arasında 90-80 ve 80-70 arasında verilerimiz olacak ancak 70′in altında verilerimiz olmayacak. Bunu verilere bakmadan anlayamayız. Şayet illaki ağaç oluşturulacaksa ve kural 0 ile 100 arasındaki notların 10′arlık dilimler halinde bölünmesi ise bu dalların da ağaçta yer alması gerekir ancak hiç veri içermeyeceği için bu dallanmalar anlamsız olacak ve budanacaktır.

C4.5 ağacı için son olarak WEKA programında J48 olarak açık kaynak kodlu bir versiyonunun yazılmış olduğunu söylemekte yarar vardır.

## **SORU-9: Graph ve RDF Veritabanları hakkında bilgi veriniz.**

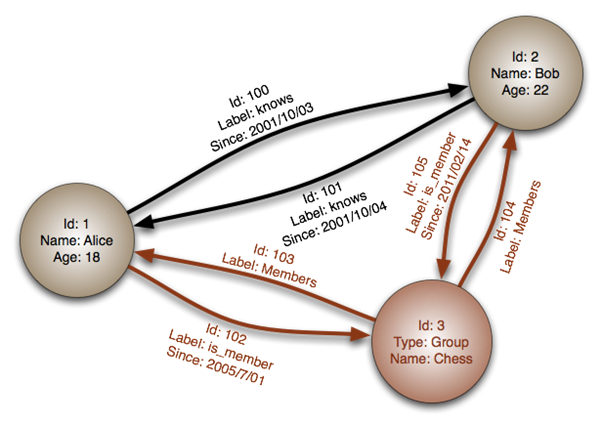
**Özet**

Bu doküman Graph ve RDF veritabanları hakkında genel bilgi sunmaktadır. Doküman 4 kısımda hazırlanmıştır. İlk kısımda Graph veritabanları ve tanımları açıklanmış, ikinci kısımda RDF veri tabanları açıklanmış, üçüncü kısımda Graph veritabanlarına örnekler verilmiş ve kısa açıklamalar ile resmi web siteleri sunulmuş ve son bölümde de RDF veritabanlarına örnekler verilmiş ve kısa açıklamalar ile resmi web siteleri sunulmuştur.

1. **Graph Veritabanları**

Bir graph veritabanı düğümler, kenarlar ve özelliklerle beraber graph yapılarını kullanarak veriyi sunar ve saklar. Tanımsal olarak bir graph veritabanı, indissiz yakınlık (index-free adjacency) sağlayan bir veri saklama sistemidir. Bu tanıma göre, her öğe (element) yakın olduğu öğeye doğrudan bir işaretçi (direct pointer) içerir ve indis aramaları (lookup) gereksizdir. Herhangi bir graph tutabilen genel graph veritabanları, “ağ veritabanları” ve “triplestores” gibi özelleştirilmiş graph veri tabanlarından farklıdır.

Graph veritabanları graph teorisine dayalıdır ve düğümler, özellikler ve kenarlar içerirler. Düğümler, nesneye yönelik programcıların aşina olduğu objelere çok benzemektedir.



Şekil 1:Düğümler

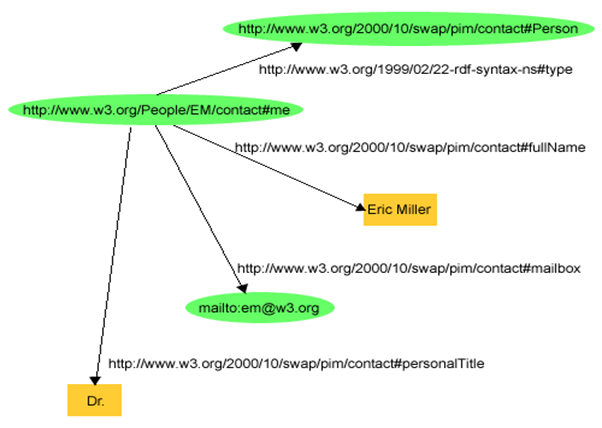
Düğümler insan, iş, hesap gibi takip edilmek istenen varlıkları (entity) temsil eder. Özellikler, düğümlerle ilişkili olan bilgilerdir. Örneğin, eğer “wikipedia” düğümlerden biriyse, “web sitesi” veya “referans materyali” gibi özelliklere bağlanabileceği gibi düğümün hangi özelliklerinin veritabanı ile uyumlu olacağına bağlı olarak, “‘w’ ile başlayan kelimeler” gibi özelliklere de bağlanabilir.

Kenarlar düğümleri düğümlere veya düğümleri özelliklere bağlayan çizgilerdir ve iki taraf arasındaki ilişkiyi temsil eder. Önemli bilginin çoğu kenarlarda tutulur. Birisi düğümlerin, özelliklerin ve kenarların bağlantılarını incelediği zaman anlamlı örüntüler ortaya çıkar. [1]

**2. RDF Veritabanları**

Aslen bir “metadata” veri modeli olarak tasarlanan RDF (Resource Description Framework), World Wide Web Consortium (W3C) spesifikasyonlarının bir ailesidir. Web kaynakları içerisinde uygulanmış olan bilginin, çeşitli sözdizimi formatlarını kullanarak modellenmesi veya kavramsal olarak tanımlanması için kullanılan bir metot olagelmiştir.

RDF veri modeli, varlık-ilişki veya sınıf-diyagramları gibi klasik kavramsal modelleme yaklaşımlarına benzemektedir çünkü kaynaklar hakkında özne-yüklem-nesne ifadesi formunda değerlendirme yapma fikrine dayanmıştır. Bu ifadeler RDF terminolojisinde “Triples” olarak yer almaktadır. Özne kaynağı, yüklem kaynağın özelliklerini ifade eder ve nesne ile özne arasında bir ilişkiyi ifade eder. Örneğin, RDF’te “Gökyüzü mavi renktedir” kavramını “triple” kullanarak temsil etmenin bir yolu: “gökyüzü”nü ifade eden bir özne, “renktedir”i ifade eden bir yüklem ve “mavi”yi ifade eden bir nesnedir. Dolayısıyla RDF, nesne tabanlı tasarımdaki klasik gösterim olan varlık-özellik-değer modelinde kullanılan nesneleri, öznelerle yer değiştirir; nesne (gökyüzü), özellik (renk) ve değer (mavi). RDF birçok serileştirme biçimine (örneğin; dosya biçimi) sahip bir soyut modeldir ve dolayısıyla a bir kaynağın veya triple’ın kodlanması, biçimden biçime farklılıklar göstermektedir.



Şekil 2: Bağlantılar

Bir RDF ifadeleri koleksiyonu özünde etiketli, yönlü (directed) bir çoklu-graph’tır. Bu yüzden RDF-tabanlı veri modeli doğal olarak belirli türdeki bilgi temsiline (knowledge representation), ilişkisel model veya diğer ontolojik modellerden daha uyumludur. Ancak pratikte RDF verisi genellikle ilişkisel veritabanı veya Triplestores (eğer her bir RDF triple’ı için bağlam da persist ediliyorsa Quadstores) adı verilen yerel temsillerle persist edilir. RDFS ve OWL’nin gösterdiği gibi, ek ontoji dilleri RDF kullanılarak inşa edilebilir. [2]

**3. Graph Veritabanı Projeleri**

Aşağıdaki listede iyi bilinen birçok graph veritabanı projesi listelenmiştir:

* AllegroGraph – ölçeklenebilir, yüksek performanslı RDF ve graph veritabanı
* Bigdata – yüksek derecede ölçeklenebilir RDF/Graph veritabanı, bir düğümde +10milyar kenar veya çok yüksek çıktı için kümelenmiş deployment
* CloudGraph – graph ve anahtar/değer çiftlerini veri saklamak için kullanan, disk ve hafıza tabanlı, tam işlemli (transactional) .NET graph veritabanı
* Cytoscape – açık-kaynak kodlu platform, open-source platform, biyo-enformatiğin sonucu
* DEX – Sparsity Technologies’ten yüksek performanslı bir graph veritabanı
* Filament – graph persistence çatısı ve seyirsel sorgu tarzına dayalı birleşmiş araç kutuları
* GiraffeDB – karmaşık semantikleri etkin ve erişilebilir bir yolla temsil edebilen, .NET framework 4.0 için güçlü bir grap veritabanı sistemi
* GraphBase – biçimlendirilebilir, dağıtık, yüksek performanslı ve zengin araç setli graph deposu
* HyperGraphDB – genelleştirilmiş hipergraph’ların desteklendiği, kenarların diğer kenarı işaret ettiği, açık-kaynak kodlu grap veritabanı an open-source (LGPL) graph database supporting
* InfiniteGraph – yüksek ölçeklenebilirlik dağıtık, bulut-uyumlu esnek lisanslara sahip ticari ürün
* Neo4j – açık-kaynak kodlu ve ücretli graph veritabanı
* OpenLink  
  Virtuoso – yüksek performanslı RDF graph veritabanı sunucusu, yerel gömülü durum olarak deploy edilebilir, tek-durum ağ sunucusu veya çok büyük derecede ölçeklenebilir sıfır-paylaşım ağ kümesi durumu
* OrientDB – yüksek performans, açık-kaynak kodlu, doküman-graph veritabanı

**4. RDF Veritabanı Projeleri**

* Oracle Database Semantic Technologies – hem bedava hem de ücretli sürüm, ileri-zincirli muhakeme (forward-chaining reasoning), ölçeklenebilir
* BigData RDF Database – hem bedava hem de ücretli sürüm, yatay ölçeklenebilir, transaction destekli, yüksek G/Ç oranları, SPARQL ve RDFS
* OWLIM – hem bedava hem de ücretli sürüm, Java ile geliştirilmiş yerel RDF motoru, Sesame ve Jena, RDFS ve OWL 2RL, en iyi ölçekleme, yükleme ve değerlendirme performansı
* 4Store – bedava, ölçeklenebilir, güvenli, hızlı, güçlü
* Mulgara – açık-kaynak kodlu, ölçeklenebilir, tamamen Java
* ViziQuer – SPARQL endpoint ontolojisini taramaya ve SPARQL sorguları oluşturmaya yarayan bir araç
* SemWebCentral – açık-kaynak kodlu semantik web araçları
* Virtuoso Universal Server – RDF veri yönetimi için SPARQL uyumlu platform, SQL-RDF entegrasyonu ve RDF tabanlı bağlı-veri deployment’ı
* ROWLEX – RDF dokümanlarını kolayca yaratmak ve göstermek için kullanılan .NET kütüphanesi ve araç kutusu, RDF tripler’larının seviyesini soyutlar ve programlama işini OWL sınıflarına ve özelliklerine yükseltir.
* StrixDB: bir RDF graph deposu, SPARQL, httpd olarak kullanılabilir

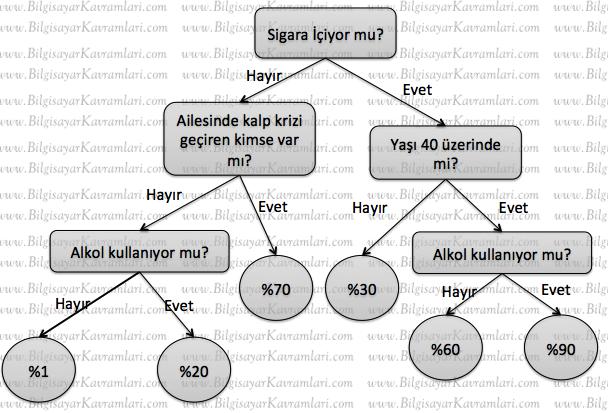
## **SORU-10: Karar Ağacı Öğrenmesi (decision tree learning) hakkında bilgi veriniz.**

Karar ağacı öğrenmesi (decision tree learning) yöntemi, makine öğrenmesi (machine learning) konularından birisidir. Literatürde karar ağacı öğrenmesinin alt yöntemleri olarak kabul edilebilecek sınıflandırma ağacı (classification tree) veya ilkelleştirme ağacı (regression tree ,tahmin ağacı) gibi uygulamaları vardır.

Karar ağacı öğrenmesinde, bir ağaç yapısı oluşturularak ağacın yaprakları seviyesinde sınıf etiketleri ve bu yapraklara giden ve başlangıçtan çıkan kollar ile de özellikler üzerindeki işlemeler ifade edilmektedir.

Bu yazı karar ağacı olarak geçen (decision tree) konusunu değil, veri madenciliği (data mining) ve makine öğrenmesi (machine learning) konularında geçen, karar ağacı öğrenmesi ile ilgilidir.

Karar ağacı öğrenmesi sırasında, öğrenilen bilgi bir ağaç üzerinde modellenir. Bu ağacın bütün iç düğümleri (interior nodes) birer girdiyi ifade eder. Örneğin aşağıda örnek bir öğrenilmiş ağaç gösterilmiştir:



Şekil 1: Hayali sonuçlar

Yukarıdaki şekil, kalp krizi geçiren kişilerin çeşitli özellikleri ele alınarak veriler üzerinden öğrenilmesi durumunda hayali olarak elde edilecek sonuçlardır. Diğer bir deyişle, iç düğümlerde (dikdörtgenler) çeşitli girdilere göre ağacın dallanması söz konusudur. Yapraklarda ise sonuç olarak elde edilen değerler gösterilmiştir.

Karar ağacı öğrenmesinde, ağacın öğrenilmesi sırasında, üzerinde eğitim yapılan küme, çeşitli özelliklere göre alt kümelere bölünür, bu işlem, özyineli olarak (recursive) tekrarlanır ve tekrarlama işleminin tahmin üzerinde bir etkisi kalmayana kadar sürer. Bu işleme özyineli parçalama (recursive partitioning) ismi verilir.

Genelde veri madenciliği sırasında verinin gelme şekli aşağıdaki gibidir:

(x,Y) = (x1,x2,x3,x4,…, Y)

Bu gösterime göre x1′den xn’e kadar olan değerler, sistemin girdileri iken, Y değeri sistemin çıktısı olarak elde edilmesi istenen değerdir. Örneğin yukarıdaki şekilde, kalp krizi geçirme oranı Y değeri olarak, “ailesinde kalp krizi geçirenlerin olup olmaması”, “kişinin yaşı”, “alkol kullanıp kullanmaması”, “sigara içip içmemesi” gibi bilgiler ise sistemin girdileri olan x değerleri olarak düşünülebilir.

Veri madenciliğinde karar ağacı öğrenmesi (decision tree learning) iki temel amaç için kullanılır. Bu amaçlar ve karar ağacı öğrenmesinin bu amaca yönelik özel isimleri aşağıdaki şekildedir:

* Sınıflandırma problemleri : Sınıflandırma ağaçları (Classification Tree) : Bir kişinin harcamalarından eğitim düzeyinin tahmini gibi, hedef kümeyi çeşitli sınıflardan birisine yerleştirmeyi amaçlayan ve sınıf tanımı yapan problemler.
* İlkelleme problemleri : İlkelleme ağaçları (Regression Trees): Sonuçta bir sınıf yerine sayısal bir değer döndüren veri madenciliği problemleri : Örneğin yukarıda verilen kalp krizi geçirme ihtimalleri gibi.

Ayrıca yukarıdaki iki terimi de içine alan bir çatı terim olarak CART (Classification and Regression Tree) terimi de kullanılmaktadır.

**Karar ağacı öğrenme algoritmaları**

Bu ağaçların çalıştırılması sırasında aşğıdaki bazı algoritmalardan yararlanılabilir (İnşaAllah vakit bulursam bunları da ayrı birer yazı halinde site üzerinden yayınlayacağım):

* Rastgele Orman (Random Forest) : Sınıflandırma işlemi sırasında birden fazla karar ağacı kullanılarak sınıflandırma değerinin yükseltilmesi hedeflenir.
* Hızlandırılmış Ağaçlar (Boosted Trees): Hem sınıflandırma (classification) hem de ilkelleme (regression) problemleri için kullanılabilen bir algoritmadır.
* Döndürme Ağacı (Rotation Forest) : Rastgele ağaca benzer şekilde birden fazla ağaç kullanılmaktadır ancak her ağaç, önce farklı bileşen analizi (Pricipal Component Analysis, PCA) kullanılarak eğitilmektedir. Bu eğitim için veri kümesinin rast gele seçilmiş bir alt kümesi kullanılmaktadır (sarnıçlama yöntemiyle).
* Ayırca aşağıdaki algoritmalar da karar ağacı öğrenmesinde kullanılmaktadır:
* ID3 algoritması
* C4.5 algoritması
* Chi-Kare Otomatik İlişki Tarayıcısı (Chi-Square Automatic Interaction Detector, CHAID) : Birden fazla seviyeye bölme işlemine izin veren bir sınıflandırma algoritmasıdır.
* MARS : Sayısal verilerin daha iyi işlenebilmesi için karar ağaçlarını iyileştiren bir yaklaşımdır.

**Karar ağacı öğrenme algoritmalarının avantajları**

* Anlaşılması ve yorumlanması basittir. Basit bir açıklama ile karar ağacı öğrenmesinin neticelerini insanlar anlayabilirler.
* Hızlı ver ön işlemesi gerektirir: Çoğu alternatif tekniklere göre çok az bir işleme ile veri kullanılabilir hale gelir. Ön işleme aşaması diğer alternatiflerine göre daha kısa ve basittir.
* Hem sayısal hem de sınıfsal verilerin işlenmesi için kullanılabilir: Çoğu makine öğrenmesi algoritması, ya sayısal uygulamalarda kullanışlı ya da sınıflandırma problemleri için kullanışlıdır. Karar ağacı öğrenmeleri ise, iki alanda da kullanılabilir.
* Beyaz kutu modelini kullanır: Yazılım mühendisliğinin (software engineering) bir yaklaşımı olan beyaz kutu modelinde, her adım görüntülenebilir ve yorumlanabilirdir. Yine bir yazılım mühendisliği uygulaması olan kara kutu yaklaşımı (black box approach) ise makine öğrenmelerinde daha çok yapay sinir ağları (artificial neural network) tarafından karşılanır. Bu yöntemde girdi ve çıktı yorumlanabilirken sistemin iç dinamiklerinin her adımda gözlemlenmesi ve yorumlanması mümkün değildir.
* Düşük hesaplama karmaşıklığına (computational complexity) sahiptir: Basit ve hızlı olmasından dolayı yüksek miktardaki veriyi kısa sürede işleyebilir ve alternatifi yöntemlere göre veri miktarı arttığında daha da tercih edilebilir olur.

**Yöntemin Kısıtları**

İyileştirilmiş (optimum) bir karar ağacı öğrenme algoritması çoğu zaman NP-Tam (NP-Complete) karmaşıklığa sahiptir. Bu karmaşıklığından sakınıldığı için çoğu karar ağacı öğrenme algoritması, sezgisel algoritmalar (heuristic algorithms) veya aç gözlü yaklaşımı (greedy approach) kullanmaktadır. Genelde bu uygulamalar, hedefe yönelik olarak, yerel olarak iyileştirilmiş (local optimum) bulmada başarılı olurken, genel iyileştirmeyi (global optimum) kaçırmaktadır.

Karar ağacı öğrenmesi ile çalışılırken kolaylıkla yapılacak bir hata, ağacı özel amaçlara yönelik olarak değiştirirken, bütün verinin özelliklerini modelleyemeyen bir ağaç ile karşılaşılabilir. Bu durum makine öğrenmesi konusunda aşırı uyum (overfitting) olarak geçmektedir. Bu durumu engellemek için budama (prunning) işlemleri yapılmaktadır.

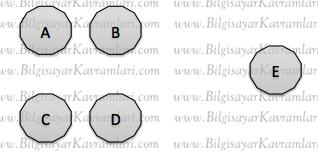
Makine öğrenmesi konularında meşhur olan ve çoğu algoritma için problem olan karmaşık konular (örneğin özel veya problemi (XOR problem) , teklik problemi (parity problem) veya çoklayıcı problemi (multiplexer) problem gibi) karar ağacı öğrenmesi tarafından da modellenmesi ve çözümlenmesi zor konulardır. Bu problemlerin çözümü ya problemin tanım alanının yeniden tanımlanması (ki bu işleme haberselleştirme (propositionalisation) ismi verilir) veya algoritmanın daha açılayıcı bir gösterime dönüştürülmesi sayesinde yapılmaktadır.

Birden fazla seviyeden oluşan sınıfsal verilerin genelde diğer verilerden yüksek sonuç vermesi söz konusudur.

## **SORU-11: Öbekağacı (dendrogram) hakkında bilgi veriniz.**

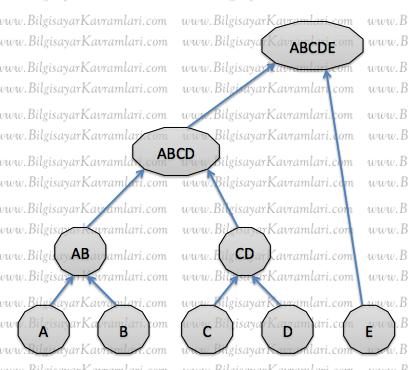
Öbek ağaçları (dendrogram) latince iki kelimenin birleşimidir. Dendron kelimesi, ağaç anlamına, gramma kelimesi ise çizim anlamına gelmektedir. Çeşitli bilgisayar bilimleri uygulamalarında kullanılırlar. Örneğin hesaplamalı biyoloji (computational biology) konusunda genler arasındaki ilişkileri modellemek için veya veri madenciliği (data mining) konusunda veri grupları arasındaki bilgileri modellemek için kullanılırlar.

Konuyu basit bir örnek üzerinden açıklayalım. Örneğin aşağıda verilen 5 veri grubunun aralarındaki mesafenin öklit mesafesi (euclidean distance) olarak hesaplandığını kabul edelim.



Şekil 1: Öklid mesafesi

Buna göre A ile B birbirne en yakın, C ile D birbirine en yakın ve E ise hepsine en uzak noktada yerleştirilmiştir. Yukarıdaki bu veri gruplarının öbek ağacını (dendrogram) çizersek aşağıdaki gibi bir şekil elde edilebilir:



Şekil 2: Öbek ağacı

Yukarıda görüldüğü üzere birbirine yakın olan gruplar önce öbeklenmiş ve en uzaktaki grup en son öbeklemiştir. Öbekleme işlemi bütün grupları kapsayacak şekilde yakınlığına göre ilişkilendirerek gitmektedir. Bu ilişkiler ikili ilişkiler (binar relation) olarak düşünülmelidir. Yani her yeni öbek, kendinden önceki en fazla iki öbeğin birleşimi olarak düşünülebilir.

Yukarıdaki şekilde aşağıdan yukarıya doğru çizilen ağaç, bazı kaynaklarda tam ters istikamette çizilmiş gösterilmektedir. Bazı kaynaklarda ise aynı ağaç, bağlantı yönleri ters çevrilerek, ikili ağaç (binary tree) şeklinde gösterilmektedir. Bütün gösterimlerdeki amaç, birbirine yakın olan grupları ağaçta daha yakın komşular olarak göstermektir.

Yukarıdaki şekli ayrıca ilişki mesafesi olarak okumak da mümkündür. Örneğin A grubu ile E grubu arasında bir ilişki kurulabilmesi için A grubu önce 2 farklı gruplamaya tabi tutulmalıdır.

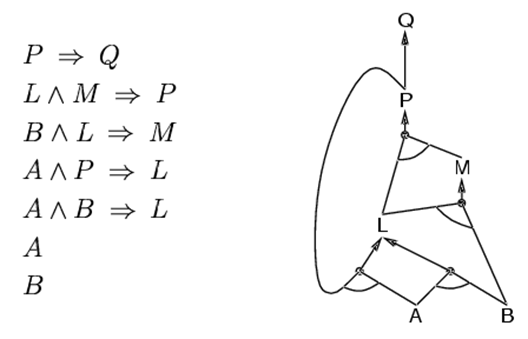
## **SORU-12: İleri ve geri zincirleme (Forward and Backward Chaining) hakkında bilgi veriniz.**

Bu yazının amacı, bilgisayar bilimlerinde, özellikle de mantıksal sistemlerin ispatında kullanılan ileri zincirleme ve geri zincirleme yöntemlerini açıklamaktır.

Yöntemin çalışması oldukça basittir. Öncelikle problem, mantık düzleminde modellenir. Buradaki mantık sistemi sonlu ispatı olan herhangi bir system olabilir. Örneğin birinci dereceden mantık (first order logic) veya daha özel olarak boole cebiri kullanılabilir.

Modelleme aşamasının ardından problemin çözümüne geçilir. İşte tam bu noktada ileri zincirleme (forward chaining) veya geri zincirleme (backward chaining) yöntemlerinden birisi seçilebilir.

Örneğin aşağıdaki mantıksal sistemi ve şekli ele alalım:



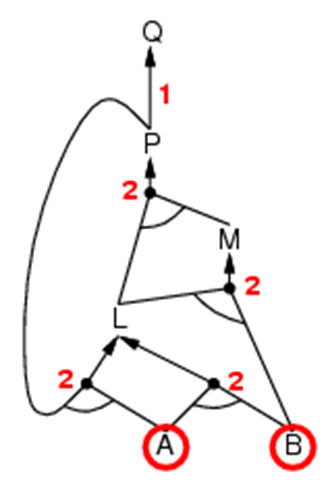
Şekil 1: Mantıksal dizilim

Sistemde görüldüğü üzere bazı mantıksal dizilimler verilmiş ve son iki satırda A ve B önermelerinin (kaziye) doğru olduğu belirtilmiştir.

Buna göre sağdaki çizim, hangi durumlarda, hangi diğer durumların doğru olacağını bu mantıksal sistemden çıkarır. Örneğin p=> q ifadesi, çizimin en tepesinde gösterilmiş ve p önermesinin (predicate, kaziyesinin) doğruluğu halinde q önermesinin de (kaziyesinin de) doğru olacağını ifade etmektedir.

Benzer şekilde, L önermesinin (kaziyesinin) doğruluğu A ve B önermelerine bağlı olduğu gibi, A ve P önermelerinin doğruluğuna da bağlanmıştır. Bu iki sistemden birisinin doğru olması sonucun doğruluğunu sağlar.

Şimdi şekilde gösterilen sistemi ileri zincirleme (forward chaining) yöntemi ile çözelim. Öncelikle sistemdeki bütün doğruluk şartlarını sayısal olarak ifade ediyoruz:

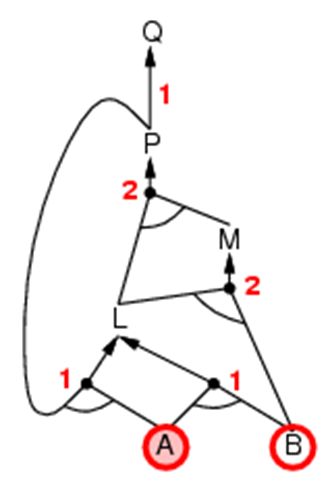


Şekil 2: Doğruluk şartları

Şekilde görüldüğü üzere bütün doğruluk şartları birer sayı ile ifade edilmiştir. Söz gelimi, M önermesinin doğruluğu L ve B önermesi gibi 2 önermenin doğruluğunu gerektirir. Bu yüzden M birleşiminde 2 sayısı bulunur. Benzer şekilde Q önermesinde bulunan 1 sayısı, sadece P önermesinin doğruluğunun yeterli olduğunu ifade etmektedir.

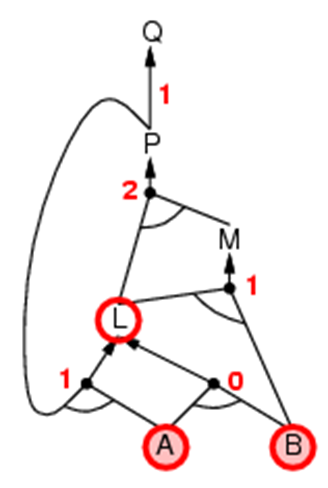
Şimdi ileri zincirleme yöntemini kullanarak sistemin doğru olduğu verilen A ve B önermelerinden itibaren çözümünü izleyelim.

Öncelikle A ve B’ye komşu olan düğümlerdeki değerleri 1′er azaltıyoruz:



Şekil 3: Zincirleme işlemi

Yukarıdaki şekilde ileri zincirleme işlemi (forward chaingin) A önermesi için çalıştırılmış olup A’nın komşularını 1 azaltmıştır. Sırada B önermesi var ve onu da çalıştıralım:

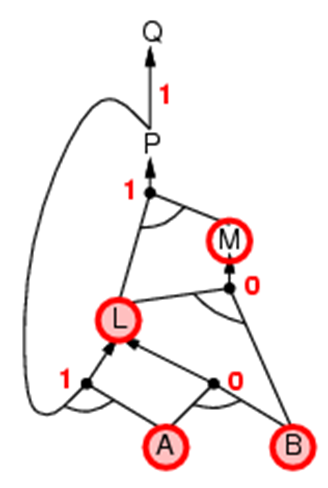


Şekil 4: Zincirleme sonucu

Görüldüğü üzere B’nin komşuları da 1 azaldığında 0 değerine sahip bir düğüm elde ettik. Bu durumda L’nin doğru olduğunu söyleyebiliriz çünkü L’nin doğru olması için gereken 2 değer de sağlandı. Yani mantıksal sistemimizde bulunan

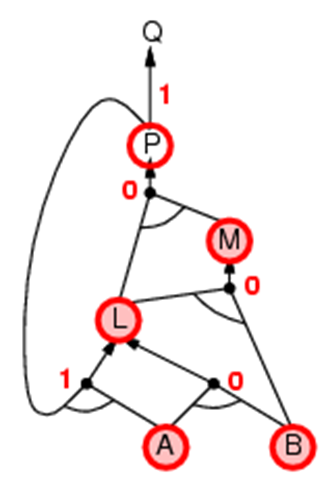
A  B  L

Satırını sağlamış olduk. Buradan doğruluğunu bulduğumuz L önermesinin komşularını 1 azaltıyoruz:



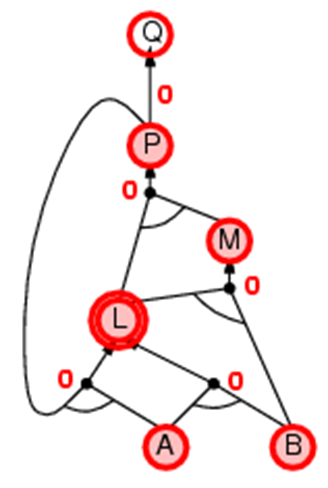
Şekil 5: İşlemler

L’nin komşularının 1 azalması sonucunda M’nin değeri 0′a inmişi oluyor ve artık M için de doğru diyebiliyoruz. Şimdi M’nin komşularını 1 azaltalım:



Şekil 6: İşlemler

Artık P için doğru sonucuna ulaştık ve P’nin iki komşusununda değerini 1 azaltarak sonucu buluyoruz:



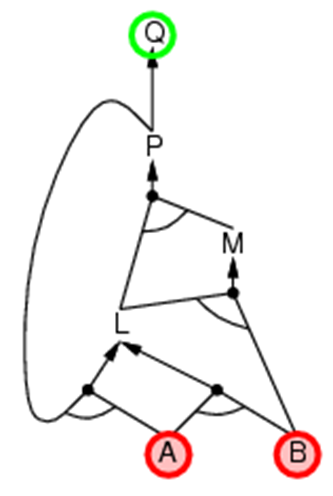
Şekil 7: İşlemler

Görüldüğü üzere zaten doğru olduğunu bildiğimiz L için tekrar doğru sonucunu bulduk ve ilave olarak Q için de doğru sonucunu bulduk.

Demek ki ilk sistem bize verildiğinde, Q’nun değeri sorulsaydı, doğru olduğunu söyleyebilirdik, ancak bunu bilgisayarın bulması için yukarıda adım adım anlatılan aşamaların tamamlanması gerekmektedir.

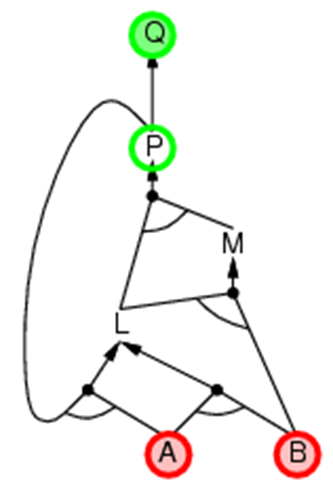
**Geri Zincirleme (Backward Chaining)**

Gelelim aynı amaç için kullanılan, yani bir mantıksal sistemi çözmek için kullanılan geri zincirleme yöntemine. İleri zincirleme yöntemine çok benzer olarak yine bir mantıksal sistem, bir şekil üzerinde gösterilebilir:



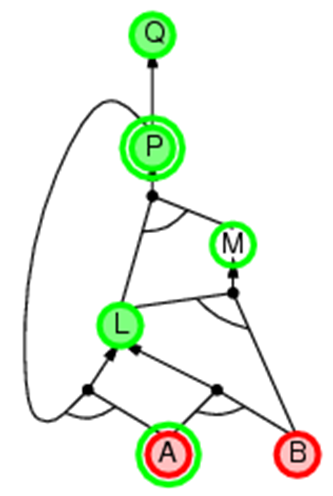
Şekil 8: İşlemler

Sistemde diyelim ki Q’nun değerini merak ediyor olalım. Bilgisayar algoritması, bu defa Q’nun değerinin doğruluğunun P’nin değerinin doğruluğuna bağlı olduğunu çözerek işe başlayacaktır. Aslında geri zincirlemede kullanılan yaklaşım, tam olarak ileri zincirlemenin tersidir. İleri zincirlemede, doğruluğunu bildiğimiz önermelerden başlanırken, geri zincirlemede, doğruluğunu aradığımız önermelerden başlıyoruz. Burada doğruluğunu aradığımız önerme Q olduğuna göre, Q’dan başlayarak sistemi dolaşacağız. İlk adımda Q’nun doğruluğu, P’nin doğruluğuna bağlıdır, o halde Q yerine P doğru mudur diye sistemi çözmeye çalışırız:



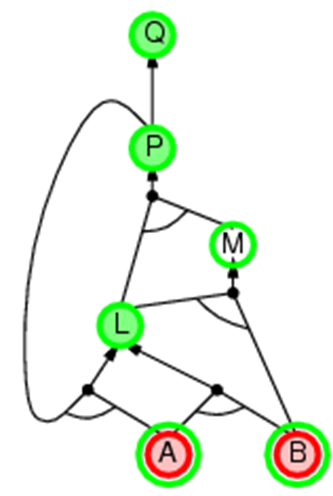
Şekil 9: İşlemler

P’nin doğruluğu, şekilde de görüldüğü üzere, L ve M’ye bağlıdır ve artık L ve M doğru mudur diye sorarız. Bunlardan birisinin yanlış olması halinde sonuç yanlış veya ikisinin de doğru olması halinde sonuç doğru olacaktır. Burada sonuç ile kastedilen P ve dolayısıyla Q’dur. Dikkat edilirse artık L ve M’ye bakarak Q’nun değerini tahmin edebiliyoruz. Devam edelim:



Şekil 10: İşlemler

L’nin doğruluğuna bakıldığında P ve A bulunmakta, aslında bu sorunun cevabını P’nin değerini bilmediğimiz için veremeyiz. Ancak burada bir tehlike bizi bekliyor, şayet doğruluğunu araştırmak için DFS (depth first search, derin öncelikli arama) benzeri bir algoritma ile ağacı (veya şekli (graph) ) dolaşıyorsak, bu durumda bir sonsuz döngüye (fasit daire) girme ihtimalimiz bulunuyor. Bunu engellemek için diyelim ki derinliği sabitledik ve L’nin doğruluğu için A ve B ikilisine bakmaya karar verdik:



Şekil 11: İşlemler

Sonuçta A ve B doğru ise L doğru demektir. O halde L doğru mu sorusunu sormayı bırakıyor ve M doğrumu A ve B doğru mu sorularını arayarak sistemi çözmeye devam ediyoruz. M’nin doğruluğu ise L ve B’ye dayanmakta, o halde bir kere daha L’nin doğruluğunu sorguluyor ve yukarıda anlatıldığı üzere bir kere daha A ve B’nin doğruluğunu sorguluyoruz. Neticede sorumuz basitçe A ve B doğru mudur şeklinde oluyor.

Verilen mantıksal sistemden de bildiğimiz üzere A ve B doğrudur, o halde Q da doğrudur diyebiliriz, çünkü sistemi buraya kadar adım adım çözdük ve neticede Q’nun doğruluğunu sorgulamanın A ve B’nin doğruluğunu sorgulamak olduğunu gördük.

Geri zincirleme (backward chaining) yaklaşımında istenirse buradan geriye dönülerek bütün sistemdeki önermelerin durumları doğru veya yanlış olarak işaretlenebilir. Ancak geri zincirleme algoritması, bu aşamada aranan Q önermesinin sonucunu bularak durabilir de. Bu iki yaklaşım arasındaki fark aslında CPS (call by passing style) ile birikimsel tarz (accumulation style) arasındaki fark gibidir.

İki yöntemde de sonuç doğru bir şekilde bulunur. Belki ufak bir fark olarak dikkat edilmesi gereken, geri zincirlemede, özel olarak aranan bir önermenin sonucuna konsantre olmamız, buna bağlı olarak da bazı büyük sistemlerde, sistemin sadece belirli bir kısmını çözüyor olmamız görülebilir. Buna mukabil, ileri zincirleme yaklaşımında, sistemin tamamı çözülmektedir.

## **SORU-13: PROLOG hakkında bilgi veriniz.**

Bu yazının amacı, PROLOG diline giriş yapmak ve basit bazı yapay zeka problemlerinin PROLOG dilinde nasıl kodlanarak çözüldüğünü göstermektir.

**Kurulum ve çalıştırma:**

Bu yazı kapsamında SWI-PROLOG programı kullanılacaktır. Programı, www.swi-prolog.org adresinden temin etmek mümkündür.

Yazı kapsamında MAC OSX üzerinde örnekler çalıştırılarak gösterilecektir ancak kurulum ve sonrasında başarılı bir çalıştırma yapılabiliyorsa hangi işletim sisteminde olduğunuzun bir önemi yoktur, bütün sistemlerde aynı PROLOG komutları çalışır.

Siteden indirilip kurulum yapıldıktan sonra,

$ /opt/local/bin/swipl

Dizini altında bulunan swipl komutu çalıştırılarak, PROLOG komut satırı açılabilir.

Program çalıştırıldığında, aşağıdakine benzer bir komut satırı ile komut girilmesini bekleyecektir:

?- \_

**Yardım**

İlk ve en önemli komutumuz olan yardım komutu ile başlayabiliriz.

?- help(help).

Bu komut, X11 ekranında, yardım dokümanını görüntüleyecektir. Basitçe herhangi bir komut hakkında bilgi alınmak istendiğinde bu komutu help fonksiyonuna parametre olarak vermek yeterlidir. Yukarıdaki kullanımda help komutunun kendisi için yardım alınmıştır.

**Çıkış**

Prolog komut satırından çıkmak için

?- halt.

Yazılması yeterlidir. Bu komut programı sonlandıracaktır.

**Basit Kaziyeler**

Aşağıdaki komutları PROLOG üzerinde çalıştıralım ve ardından burada yapılan işlemleri yorumlayalım.

Sistemimizde kayıtlı olan bir dosyayı program tarafından açıp çalıştırılır bir şekilde yüklemek için dosya ismi köşeli parantezler içerisinde verilmelidir:

?- [‘ilkdosya.pl'].

Yukarıda, bilgisayarımızda bulunan “ilkdosya.pl” isimli dosyayı sisteme tanıtarak yükledik. Bu dosyada bulunan tanımlar da otomatik olarak yüklenmiş ve sorgulanmaya / çalışmaya hazır hale getirilmiş olacaktır. Ayrıca bu dosyada bulunan genel hatalar, yükleme sırasında ekrana basılır.

**PROLOG ile Faktöriyel Kodu**

PROLOG dili, yapısal olarak özyineli (recursive) bir yapıdadır ve döngülerin yerine özyineli fonksiyon (recursive function) kullanılması gerekir. Bu kullanımı göreceğimiz en basit uygulamalardan birisi olan faktöriyel kodunu yazıp kodlamaya çalışalım.

Herhangi bir editör açılarak (ben tercihen vi kullanıyorum, siz de istediğiniz bir editör ile dosyayı açıp içeriğini ) aşağıdaki şekilde yazabilirsiniz:

factorial(0,1).

factorial(A,B) :-

             A > 0,

             C is A-1,

           factorial(C,D),

           B is A\*D.

Ardından aşağıdaki şekilde dosyamızı sisteme yükleyelim:

?- ['ilkdosya.pl'].

% ilkdosya.pl compiled 0.00 sec, 732 bytes

true.

Komut satırında dosyamızın yüklendiği ve yükleme süresi ve dosya boyutu gösterilmektedir. Son satırda yer alan true bilgisine kadar bir hata bulunmamış olması, dosyamızda bir hata olmadığını gösterir. Şimdi artık ilk fonksiyonumuzu kullanmaya başlayabiliriz.

?- factorial(5,X).

X = 120 ;

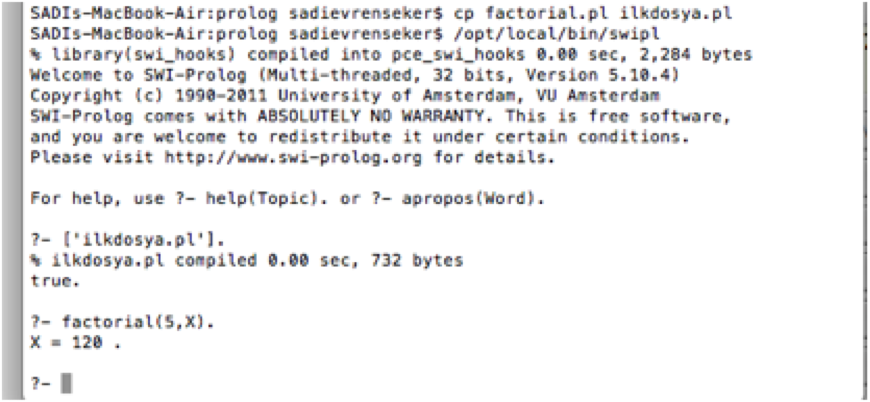
false.

Komut satırında, factorial(5,X) komutu verilerek, 5! Değerinin X değişkenine döndürülmesi istenmiştir. Burada öğrenilen birkaç önemli noktayı belirtelim. Birincisi X ile gösterilen değer bir değişkendir ve PROLOG dünyasında buna unbounded variable (bağlanmamış değişken) ismi verilir. Basitçe Prolog kodlarındaki bütün büyük harf ile başlayan ifadeler birer değişkendir ve ayrıca bir değişken tanımı yapılmaz. İkinci bir nokta, Prolog dilinde kod yazılırken girilen her satır bir emirdir ( declaration) ve yapısal olarak Prolog dili, haber mantığını (predicate calculus) barındırır. Buna göre yüklü olan dosyada bulunan komutlar yukarıdan aşağıya doğru çalıştırılırken üstte bulunan satır, alttakine gore öncelikli olur ve aynı zamanda birden fazla satırda emirin tanımlanması durumunda, sırasıyla bu emirler işlenir.

Örneğin yukarıdaki kodda, ilk satırda bulunan factorial (0,1). İfadesi, 0! = 1 anlamındadır ve bir şekilde birisi bize sıfır faktöriyeli sorarsa, ikinci değer olarak 1 sonucunu döndürmeyi gerektirir.

İkinci satırdan sonra başlayan tanımda ise, sırasıyla A>0 kontrolü yapılmış, ardından C değişkenine değer olarak A-1 değeri konmuş ve yeni bulunan C değerinin faktöriyeli hesaplanarak, B sonucuna hesaplanan bu C faktöriyel değeri ile A’nın çarpımı eklenmiştir.

Yukarıdaki şekilde kodumuzu ilkdosya.pl  dosyasına yazıp kaydettikten sonra prolog komut satırında dosyamızı yükleyip çalıştırabiliriz:

[](http://www.bilgisayarkavramlari.com/wp-content/uploads/Untitled2.png)

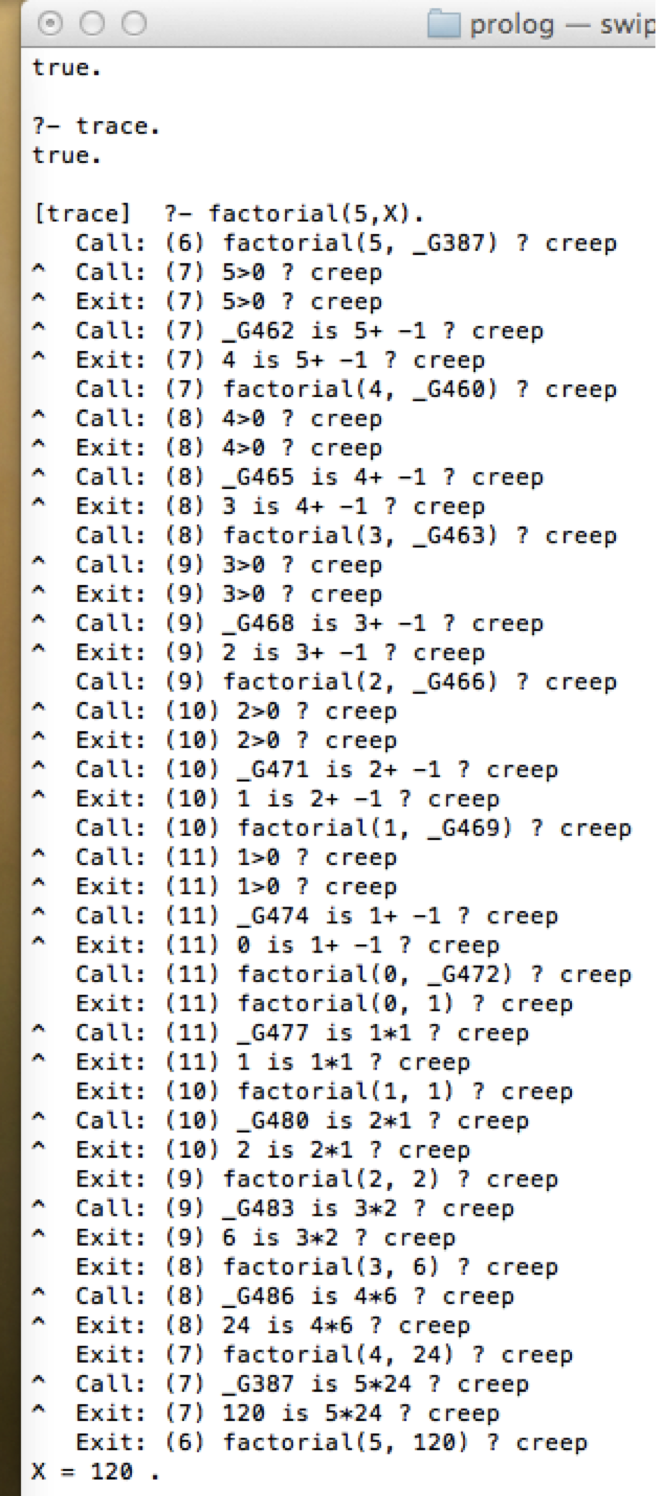
Şekil 1: ilkdosya.pl

Yukarıda görüldüğü üzere faktöriyel kodumuz çalışmış ve sonucu doğru şekilde bulmuştur.

Kodun çalışmasını takip etmek için trace komutu kullanılabilir. Basitçe komut satırında:

trace.

Yazıldıktan sonra çalışmaya başlar ve bu çalışmadan sonra çağrılan fonksiyonların detayları ekrana basılır:

[](http://www.bilgisayarkavramlari.com/wp-content/uploads/Untitled1.png)

Şekil 2: factorial fonksiyonu

Görüldüğü üzere factorial fonksiyonumuzun her adımı ekrana basılmıştır.

**Hanoi Kuleleri (Towers of Hanoi)**

İkinci bir uygulama olarak çok klasik bilgisayar bilimleri problemlerinden olan Hanoi Kulelerini ( tower of hanoi ) kodlamaya çalışalım.

Oyunun kuralı gayet basit. Başlangıçta bir çubukta dizili olan bütün diskler her adımda tek bir disk hareket ettirerek diğer iki çubuktan birisine taşınacaktır. Bu taşıma işlemleri sırasında büyük bir disk, küçük olan bir diskin üzerine gelmeyecektir.

Problemin çözümünde kullanacağımız kod aşağıdaki şekildedir:

SADIs-MacBook-Air:prolog sadievrenseker$ vi hanoi.pl

oyna(1,X,Y,\_) :-

    write(X),

    write(‘ en üstteki diskini ‘),

    write(Y),

    write(‘ oynat ‘),

    nl.

oyna(N,X,Y,Z) :-

    N>1,

    M is N-1,

    oyna(M,X,Z,Y),

    oyna(1,X,Y,\_),

    oyna(M,Z,Y,X).

Kod örneği yukarıda verilmiştir. Kodumuzun çalışmasını görüp ardından nasıl çalıştığını açıklamaya geçebiliriz.

?- oyna(4,a,b,c).

a en üstteki diskini c oynat

a en üstteki diskini b oynat

c en üstteki diskini b oynat

a en üstteki diskini c oynat

b en üstteki diskini a oynat

b en üstteki diskini c oynat

a en üstteki diskini c oynat

a en üstteki diskini b oynat

c en üstteki diskini b oynat

c en üstteki diskini a oynat

b en üstteki diskini a oynat

c en üstteki diskini b oynat

a en üstteki diskini c oynat

a en üstteki diskini b oynat

c en üstteki diskini b oynat

true .

?-

Yukarıdaki çalışmadan da görüldüğü üzere ilk baştaki 4 diskin dizili olduğu a çubuğundan bütün diskler sırasıyla b çubuğuna taşınmıştır.

Kodumuz, doğru sıra ile oynama yaklaşımı ile kodlanmıştır. Hanoi kulelerinin farklı çözümleri bulunur. Örneğin geri izleme algoritması (backtracking) kullanarak kaba kuvvetle (brute force) bütün ihtimallerin denendiği ve başarısızlık olması halinde en son iyi duruma geri dönen yaklaşımları kodlamak da mümkündür.

Bizim kodlamamızda ise her seferinde doğru çubuktan doğru diskin oynatılması esası kullanılmıştı. Bu yüzden hiç geri oynama veya yapılan hamleden vaz geçme olmamıştır.

Bu oynama esasını daha iyi anlatabilmek için daha ufak bir örnek üzerinden çalışmayı gösterelim :

?- oyna(2,a,b,c).

a en üstteki diskini c oynat

a en üstteki diskini b oynat

c en üstteki diskini b oynat

true

Görüldüğü üzere iki disklik bir dizilimde, c çubuğu geçici olarak kullanılmakta, esas büyük diskin b çubuğuna oynanması sırasında, küçük diskleri depolamak için kullanılmaktadır.

Bu durum disk sayısı artsa da değişmez. En büyük diskin a çubuğundan b çubuğuna oynanması sırasında,  diğer bütün küçük diskleri depolamak için c çubuğunu kullanırız. Buradan anlaşılacağı üzere ikinci büyük diskin c çubuğuna oynaması gerekir. Bunun için de b çubuğunu geçici depolama için kullanırız. Bu yaklaşım en büyük diskten en küçüğüne kadar aynı mantıkla işleyerek devam eder.

**8 Vezir Problemi (8 Queens Problem)**

Yeni bir problem olarak klasik sorulardan birisi olan 8 vezir problemini ( eight queens problem) ele alalım. Problem basitçe bir satranç tahtasına 8 veziri, birbirini yemeden nasıl yerleştireceğimizdir.

Problemin çözümü olan PROLOG kodu aşağıda verilmiştir.

perm([X|Y],Z) :- perm(Y,W), birinial(X,Z,W).

perm([],[]).

birinial(X,[X|R],R).

birinial(X,[F|R],[F|S]) :- birinial(X,R,S).

cozum(P) :-

     perm([1,2,3,4,5,6,7,8],P),

     birlestir([1,2,3,4,5,6,7,8],P,S,D),

     hepsifarkli(S),

     hepsifarkli(D).

birlestir([X1|X],[Y1|Y],[S1|S],[D1|D]) :-

     S1 is X1 +Y1,

     D1 is X1 – Y1,

     birlestir(X,Y,S,D).

birlestir([],[],[],[]).

hepsifarkli([X|Y]) :-  +member(X,Y), hepsifarkli(Y).

hepsifarkli([X]).

Kodun çalışan örneği aşağıdaki şekildedir:

?- cozum(P).

P = [5, 2, 6, 1, 7, 4, 8, 3] ;

P = [6, 3, 5, 7, 1, 4, 2, 8] ;

P = [6, 4, 7, 1, 3, 5, 2, 8] ;

P = [3, 6, 2, 7, 5, 1, 8, 4] ;

P = [6, 3, 1, 7, 5, 8, 2, 4] ;

P = [6, 2, 7, 1, 3, 5, 8, 4] ;

P = [6, 4, 7, 1, 8, 2, 5, 3] ;

P = [3, 6, 2, 7, 1, 4, 8, 5] ;

P = [6, 3, 7, 2, 4, 8, 1, 5] ;

…

Yukarıdaki örneklerde görüldüğü üzere her kolon için, o kolondaki kaçıncı satıra vezir yerleştirilebileceği ve bu yerleştirme işlemine göre bütün kuralları sağlayan bir dizilimin nasıl elde edileceği listelenmiştir. Problemin birden fazla çözümü olduğu için her satır, çözümlerden birisini göstermektedir. Örneğin ilk çözüm satırını ele alırsak, aşağıdaki gibi bir satranç tahtası elde ederiz.

P = [5, 2, 6, 1, 7, 4, 8, 3] ;

Kodda, kabaca bütün ihtimalleri deneyen bir permütasyon algoritması geliştirilmiştir. Ardından koşulların sağlanıp sağlanmadığı kontrol edilmiş ve buna göre koşulları sağlayan ihtimaller sonuç olarak döndürülmüştür.

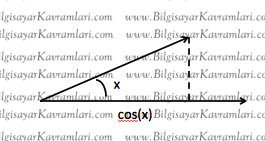
Algoritma, bir arama algoritması (search algorithm) şeklinde probleme yaklaştığı için, problemin çözümü sırasındaki bütün ihtimaller sadece bir kere denenecek ve denenen bir ihtimal tekrar etmeyecektir. Bu deneme sırası, permütasyon fonksiyonu ile sağlanır.

## **SORU-14: SimHash (Benzerlik Özeti) hakkında bilgi veriniz.**

Bilgisayar bilimlerinde, özellikle metin işlemenin yoğun olduğu, arama motoru gibi uygulamalarda dosyaların veya web sitelerinin birbirine olan benzerliğini bulmak için kullanılan bir algoritmadır.

Algoritmaya alternatif olarak klasik hash fonksiyonları kullanılabilir. Yani, örneğin iki sayfasnın ayrı ayrı hash değerleri alınıp bu değerleri karşılaştırmak mümkündür. Ancak simhash algoritması, bu yönteme göre daha fazla hız ve performans sunar.

Sim hash algoritması, iki dosyayı birer vektör olarak görür ve bu vektorler (yöney, vector) arasındaki cosinüs (cosine) bağlantısını bulmaya çalışır.



Şekil 1: Vektör

Yukarıdaki şekilde temsil edildiği üzere iki dokümanın ayrı ayrı birere vektör olması durumunda, aralarında cos (x) olarak gösterilen bir açı ile bağlantı bulunması mümkündür.

Algoritma, öncelikle işlediği metindeki kelimelerin ağırlıklarını (weight) çıkarmakta ve buna göre kelimeleri sıralamaktadır.

Sıralanan her kelimeye, b uzunluğunda, yegane (unique) değer döndüren bir fonksiyon kullanılır. Örneğin her kelime için farklı bir hash sonucu döndüren fonksiyon kullanılır.

b boyutundaki bir vektörün ağırlık değeri hesaplanırken, her kelimedeki 1 değeri için +1 ve 0 değeri için -1 değeri ağırlığa eklenir.

Son olarak üretilen ağırlık vektöründeki + değerler 1, 0 ve – değerler ise 0 olarak çevirilir.

**Örnek**

Yukarıdaki algoritmanın çalışmasını bir örnek üzerinden anlatalım. Algoritmanın üzerinde çalışacağı metin aşağıdaki şekilde verilmiş olsun:

www bilgisayar kavramları com bilgisayar kavramlarının anlatıldığı bir bilgisayar sitesidir ve com uzantılıdır

Yukarıdaki bu metni, algoritmanın anlatılan adımlarına göre işleyelim:

İlk adımımız, algoritmadaki kelimelerini ağırlıklarının çıkarılmasıdır. Bu adımı çeşitli şekillerde yapmak mümkündür ancak biz örneğimizde kolay olması açısından kelime frekanslarını (tekrar sayısı, frequency) kullanacağız. Buna göre metindeki kelimelerin tekrar sayılarına göre sıralanmış hali aşağıda verilmiştir:

bilgisayar 3 com 2 kavramları 1 kavramlarının 1 anlatıldığı 1 bir 1 www 1 sitesidir 1 ve 1 uzantılıdır 1

Yukarıda geçen her kelime için bir parmak izi (fingerprint) değeri üretiyoruz. Bu değerin özelliği, kelimeler arasında yegane (unique) bir değer bulmaktır. Bu değer, herhangi bir hash fonksiyonu üzerinden de üretilebilir. Biz örneğimizde kolalık olması açısından her kelime için rast gele bir değer kendimiz atayacağız. Ancak gerçek bir uygulamada rast gele değerlerin kullanılması mümkün değildir. Bunun sebebi, aynı kelimenin tekrar gelmesi halinde yine aynı değerin üretilmesi zorunluluğudur. Bu yazıdaki amaç algoritmayı anlatmak olduğu için birer hash sonucu olarak rast gele değerler kullanılacaktır.

bilgisayar 10101010 com 11000000 kavramları 01010101 kavramlarının 10100101 anlatıldığı 11101110 bir 01011111 www 11110001 sitesidir 10101110 ve 00001111 uzantılıdır 00100010

3. adımda, yukarıdaki değerleri topluyoruz. Toplama işlemi sırasında 1 değerleri için +1 ve 0 değerleri için -1 alıyoruz.

10101010  
11000000  
01010101  
10100101  
11101110  
01011111  
11110001  
10101110  
00001111  
00100010  
——–  
2 0 2 -4 0 2 2 0

Son olarak, yukarıdaki değerleri ikilik tabana çeviriyoruz: 10100110 bu değer bizi simhash sonucumuz olarak bulunuyor.

Örneğin yeni bir dosyayı daha işlemek istediğimizde, bu dosyadaki kelime yoğunluğuna göre yukarıda bulduğumuz simhash değerine yakın bir değer çıkmasını bekleriz.

Diyelim ki yeni bir dosyada da sadece “bilgisayar kavramları com” yazıyor olsun. Bu yazının sim hash değerini bularak karşılaştırmaya çalışalım:

bilgisayar 10101010 com 11000000 kavramları 01010101

10101010  
11000000  
01010101  
———  
1 1 -1 -1 -1 1 1 1

Değerin ikilik tabana çevrilmiş hali : 11000111

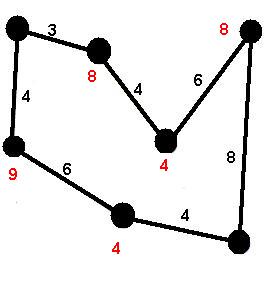
Orjinal dokümandan çıkardığımız simhash değeri ile farklı olan bit sayısı 3′tür. Bunun anlamı yukarıdaki bilgisayar kavramları com yazısının orjinal yazıya 3 mesafesinde yakın olduğudur.

## **SORU-15: Sezgisel Fonksiyonlar (Heuristic Functions) hakkında bilgi veriniz.**

Bu yazının amacı, sezgisel algoritmalar (heuristic algorithms) tarafından kullanılan sezgisel fonksiyon (heuristic function) kavramını açıklamak ve bazı zaviyelerden tasnif etmektir.

1. Baskınlık Özelliği (Dominance)  
2. Tutarlılık Özelliği (Consistent, Monotone)  
3. Makbul Özelliği (Admissible)

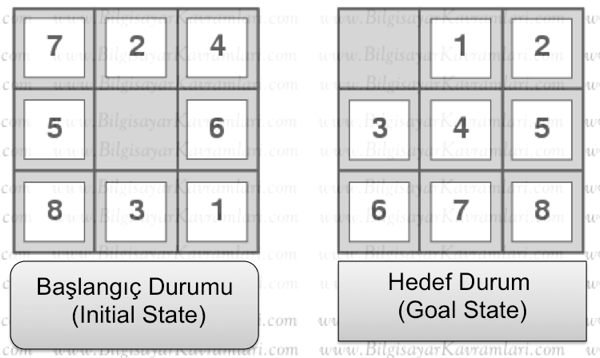
Sezgisel algoritma, en basit anlamda, bir sezgisel algoritmanın, problem bağımlı olarak bazı tahminlerde bulunmasıdır. Örneğin daha önce yazdığım A\* (A yıldız, A Star) algoritmasında, bir yol bulma örneği vermiştim. Bu algoritmadaki amaç, hedef düğüme (node) en kısa ulaşan yolu bulmaktı. Yollar karışık olmasına karşılık, hedefe giden kuş uçuşu mesafeyi sezgisel bir fonksiyon olarak kabul etmiştik:



Şekil 1: Düğümler arası mesafe

Yukarıdaki şekilde, siyah renkle yazılı olan sayılar, düğümler arası mesafeyi gösterirken, düğümlerin üzerinde kırmızı renkle yazılan sayılar, o düğümün, hedefe olan kuş uçuşu mesafesini vermektedir. Bu durumda en kısa yolu bulmayı hedeflediğimiz bu problemde, sezgisel olarak bir fonksiyon tanımlamış oluyoruz ve problem hakkında bir bilgiyi, problemin çözümünde avantaj sağlaması için kullanıyoruz.

Farklı bir problem ve farklı bir fonksiyon tanımı yaparak konuyu daha iyi izah etmeye çalışalım. Örneğin 8 bulmacası (8 puzzle) olarak bilinen, aşağıdaki klasik oyunu ele alalım.



Şekil 2: Durumlar

Yukarıdaki temsili resimde görüldüğü üzere, 8 bulmacasının bir başlangıç durumundan bir hedef durumuna kadar her adımda bir kutuyu boş olan hücreye kaydırmak suretiyle oynanması amaçlanmaktadır. Amacımıza yönelik olarak bir bilgisayar programı geliştirmek için sezgisel algoritmalardan (heuristic algorithms) faydalanmak istiyoruz. Bu durumda kullanabileceğimiz sezgisel algoritmalarda en önemli etkenlerden birisi de sezgisel fonksiyonumuz olacaktır. Yani probleme has bir fonksiyon geliştireceğiz ve bu fonksiyonla her adımda karar vermeye çalışacağız.

Bu aşamada iki fonksiyondan birisi önerilebilir:

* Taşların kaç tanesinin yerinde olmadığı
* Taşların bitiş durumunda olması gereken yere olan uzaklığı (kare mesafesi, manhattan distance)

Yukarıdaki iki farklı fonksiyon için de, yine yukarıda verilen şekildeki mesafeleri hesaplayalım.

ilk fonksiyonumuz olan, yanlış yerdeki taş sayısını hesaplamak için, yerinde olmayan taşları listeleyelim: 1,2,3,4,5,6,7,8 taşlarının tamamı hedef durumda olması gereken yerde değil. Burada boşluk hücresinin konumunu da sayarsak fonksiyon değeri olarak 9 buluruz.

İkinci fonksiyonumuza göre taşların, hedef konumuna olan mesafelerini sırasıyla listeleyelim:

1. 3
2. 1
3. 2
4. 2
5. 2
6. 3
7. 3
8. 2

Yukarıda, her taşın hedef konumuna olan mesafesi verilmiştir. Daha iyi anlaşılması açısından, örneğin 1 numaralı taş başlangıç durumunda sağ alt köşedeyken, bitiş durumunda üst ortaya gelmesi isteniyor. Bu taşın, tahtada hiç başka taş olmasaydı, başlangıç durumundan hedef durumuna gelebilmesi için yapılan hamle sayısı 3′tür. Yukarıdaki listede de her taş için bu değer sırasıyla yazılmıştır.

Fonksiyonumuzun toplam değeri : 3+1+2+2+2+3+3+2 = 18 olarak bulunur

Yukarıda, aynı problem için iki farklı sezgisel fonksiyon örneği verdik. Şimdi bu iki fonksiyonu karşılaştırarak özelliklerini açıklamaya çalışalım

**1. Baskınlık ( Dominance ) Özelliği**

Bu özellik, aynı problemin, aynı algoritma ile sezgisel olarak çözümü sırasında birden fazla fonksiyon bulunuyorsa, bu fonksiyonlar arasında seçim yapma kriterini belirler. Buna göre bir fonksiyonun alabileceği değer kumesi (fonksiyonun menzil uzayı (range) ) ne kadar geniş ise o kadar baskın fonksiyondur ve o derecede tercih edilmelidir.

Örneğin yukarıda, 8 bulmacası için iki farklı fonksiyon tanımı yaptık. Bu fonksiyonlardan ilki olan, yanlış yerdeki taş sayısı, en fazla 9 değeri alabilir. Şayet taşlardan bir tanesi doğru yerdeyse, fonksiyon değeri düşerek 8 değerine sahip olacaktr. Dolayısıyla fonksiyonun tanımlı olduğu, ve sonuç olarak döndürebileceği değerler 0 ile 9 arasındadır.

İkinci fonksiyonumuz olan manhattan mesafesi ise, taşların azami mesafede olması durumunda her taş için ayrı ayrı olarka 4 değerine sahip olabilir. (hedef durumunun (goal state) saat yönünde 4 kere çevrildiğini düşünün) ve 8 taşımız olduğu için 4 x 8 = 32 değeri, fonksiyonun döndürebileceği azami değerdir. Dolayısıyla fonksiyon 0 ile 32 arasındaki herhangi bir değeri döndürebilir.

İki fonksiyon kıyaslandığında, ilk fonksiyonun 9 ihtimalden birisini döndürebildiği, ikinci fonksiyonun ise 32 ihtimalden birini döndürebildiği görülmektedir. Bu durumda, ikinci fonksiyon, ilkine nazaran baskın fonksiyon olmakta ve ikisinden birisinin tercih edilmesi gerektiğinde, ikincisinin tercih edilmesi daha doğru olmaktadır.

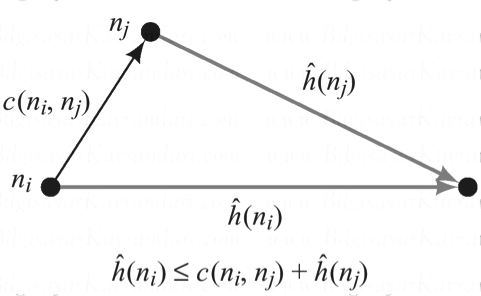
Yukarıdaki baskınlık durumunun daha iyi anlaşılması için iki fonksiyon arasındaki farkı anlatmaya çalışalım. Çok kaba anlamda, ikinci fonksiyon 32 farklı değer döndürürken, ilk fonksiyon 9 farklı değer döndürebilmektedir. Bunun anlamı, ikinci fonksiyonun ayırt ettiği bazı durumları, ilk fonksiyonun ayıramamasıdır. Örneğin ilk fonksiyondaki her değer için ikinci fonksiyonda 3-4 farklı değer bulunabilmektedir. Diğer bir deyişle, 9 ihtimalden bir tanesi, 32 ihtimalden bir kaç tanesine karşılık gelmekte, bu bir kaç farklı durum da fonksiyonun sonuca ne kadar yaklaşıldığını ölçmede ayırım oluşturmasını sağlamaktadır.

Daha net bir şekilde farkını anlatacak olursak, bulmacanın verildiği şekildeki, 8 ve 5 numaralı karolar, başlangıç durumunda yer değiştirmiş olsaydı, ilk fonksiyonumuza göre, yerinde olmayan taş sayısı yine 9 olarak bulunacaktı. Ancak ikinci fonksiyonumuzda 18 yerine 20 değeri bulunmuş olunacaktı. Bu değişim, ilk fonksiyona göre karar vermemize imkan tanımazken, ikinci fonksiyon, aralarında seçim yapma imkanı sunacaktı.

**2. Tutarlılık (Consistent, Monotone)**

Tutarlılık kavramını anlamak için üçgen eşitsizliği halini anlamamız gerekir. Çok basit anlamda, bir üçgenin üç kenarı a b ve c olmak üzere, herhangi iki kenarının toplamı, üçüncü kenardan büyük olmalıdır. Bu hale üçgen eşitsizliği hali denir ve tersten okunursa, herhangi bir kenarın, diğer iki kenarın toplamından küçük olması olarak yorumlanabilir.

Şimdi bu çok basit kavram olan üçgen eşitsizliğinin sezgisel fonksiyonla nasıl bir ilişkisi var diye sorabilirsiniz. Tutarlılık durumu, gayet basittir aslında, basitçe bir arama şeklinde (graph) herhangi bir düğümün aldığı sezgisel fonksiyon değeri, çocuğunun aldığı sezgisel foknsiyon değeri ile toplandığında çıkan sonuç çocuğunun çocğuna doğrudan giden değerden daha büyük olmalıdır. Durumu aşağıdaki şekilde görsel olarak ifade edelim:



Şekil 3: Düğüm

Yukarıdaki şekilde görüldüğü üzere en sağdaki düğüme giden ve ni düğümünden başlayan iki yol bulunmaktadır. Bu yollardan birisi doğrudan gitmekte olup, ağırlık değeri (weight) h(ni) olarak ifade edilmiştir. İkinci yol ise nj üzerinden geçen yol olup, önce c(ni, nj) (ni ile nj arasındaki gerçek mesafe (sezgisel değil) ) mesafesi gidilmekte ardından nj için sezgisel bir değer bulunmaktadır. İşte üçgen eşitsizliği, bu iki değer arasında, ikincisinin daha büyük olmasını gerektirir. Ki bu durum, yukarıdaki şekilde de aşağıdaki şekilde ifade edilmiştir:

h(ni) <e; c(ni, nj) + h(nj)

Şayet bir sezgisel fonksiyon yukarıdaki bu şartı her durum için sağlıyorsa tutarlı (consistent) sağlamıyorsa tutarsız (inconsistent) olarak tasnif edilebilir.

Fonksiyonların, tutarlı olması tercih edilir bir durumdur, bunun sebebi bir düğümden diğerine gidildiğinde ödenen maliyetin sezgisel fonksiyona yansımasının, sonucu bulmada etkisi olmasıdır. Şayet bir fonksiyon tutarsız ise, mesafe giderken harcanan maliyetin, sezgisel fonksiyona yansımadığı, en az bir durum var demektir ki bu da problemin çözümünde sorunlarla karşılaşılmasını (en azından tutarlı bir fonksiyona göre daha geç sonucun bulunmasını) beraberinde getirir.

Tutarlı bir fonksiyonun, aynı zamanda tektonlu (monotone) olarak isimlendirilmesinin sebebi ise, yukarıda izah edildiği üzere bir fonksiyonun tutarlı olması halinde, her düğümün tek bir kere ziyaret edilmesidir. Şayet bir fonksiyon tutarsız ise (inconsistent) bu durumda, aynı düğüm, arama işlemi sırasında, birden fazla kere ziyaret edilebilir. İşte bu anlamda tutarlı fonksiyonlar aynı zamanda tektonlu (monotone) olarak isimlendirilir.

**3. Makbul (Macaiz, Kabul Edilebilir, Admissable)**

Literatürde admissable olarak geçen durumdur. Bir sezgisel algoritmanın kabul edilebilir olup olmadığını (makbul) belirler. Makbul, hali, kısaca bir sezgisel fonksiyonun, herhangi bir mesafe için, gerçek mesafeden her zaman daha küçük değer vermesidir. Örneğin herhangi bir düğüm n için gerçek mesafe f(n) iken sezgisel fonksiyon olan h(n) değeri, her zaman için h(n) <e; f(n) şartını sağlamalıdır

Şayet bütün düğümler, yukarıdaki bu h(n) <e; f(n) şartını sağlıyorsa, bu sezgisel fonksiyona makbul (admissible) ismi verilebilir. Şayet sağlamayan tek bir durum varsa, bu sezgisel fonksiyonun makbul olmadığını (gayri makbul, non-admissible) söyleyebiliriz.

Yukarıdaki bu yaklaşımın sebebi, gerçek değerden daha yüksek bir sezgisel değer elde edildiği anda, bu sezgisel değerin, en iyi çözüm olan gerçek değeri perdelemesidir. Yani bir sezgisel algoritmada, makbul olmayan bir sezgisel fonksiyon kullanılırsa, bu fonksiyon için, sezgisel algoritmamız en iyi çözümü bulamayabilir.

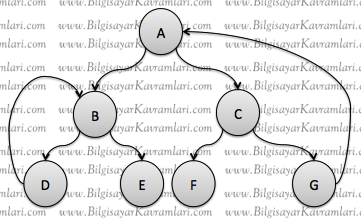
## **SORU-16: Yinelemeli Derinlik Araması (Iterative Deepining Search) hakkında bilgi veriniz.**

Bilgisayar bilimlerinin çeşitli alanlarında (örneğin yapay zeka, veri yapıları veya şekil kuramı (graph theory) gibi) kullanılan arama algoritmalarından birsidir.

Algoritma, derin öncelikli drama (depth first search) üzerine kurulu olduğu için, literatürde “iterative deepening depth first search (yinelemeli derinleşen, derin öncelikli arama)” olarak da geçmektedir.

Algoritma basitçe derinlik değerini bir değişkende tutmakta ve bu değeri her adımda arttırmaktadır.  
Yineleme yapısı (iteration) basit bir döngü (loop) olarak düşünülebilir ve her adımda derinliğin, bir döngü değişkeni (loop variable) gibi düşünülerek derinleştiği kabul edilebilir.

Örneğin aşağıdaki şekli (graph) ele alalım:



Ağaçta görüldüğü üzere iki adet döngü (cycle) bulunmaktadır. Iterative Deepening search algoritmasını bu ağaç üzerinde çalıştıracak olursak, algoritma öncelikle derinlik değerini (bundan sonra d değişkeni (variable) ile ifade edilecektir) 0′dan başlatarak her adımda bir ilerletecektir.

d=0 için arama:  
A  
d=1 için arama:  
A (tekrar A değerine bakar) B C (sanki ağacın en üstteki üç düğümü, şekilde gösterildiği gibi bağlanmış kabul edilebilir, daha alttaki derinliklerde bulunan DEFG düğümlerine hiç bakamaz)  
d=2 için arama:  
A B D E C F G (Bu aramada sanki sondaki daireler (cycle) bulunmuyormuş gibi kabul edilebilir, çünkü belirtilen derinlik bu dairelere kadar inemez)  
d=3 için arama:  
A B D B E C F G A  
d=4 için arama:  
A B D B D E B C F G A B C

Yukarıdaki arma işlemi, derinlik arttıkça devam etmektedir.

Arama algoritmalarının değerlendirildiği bazı kriterler bulunmaktadır. Buna göre IDDFS (iterative deepening depth first search) algoriması “tam” algoritma olarak kabul edilebilir.  
**Tamlık teriminin (completeness)** anlamı aşağıdaki şekilde tanımlanabilir:

Bir arama algoritması, bütün düğümleri dolaşarak aranan düğümü bulmayı garanti ediyorsa bu algoritmaya tam arama algoritması ismi verilir.

Örneğin yine yukarıdaki şekil için klasik derin öncelikli arama (depth first search) algoritmasını ele alsaydık, bu algoritmanın tam olmadığını söyleyebilirdik. Bunun sebebi DFS algoritmasının dolaşması sırasında, aşağıdaki döngüye girmesidir:

A B D B D B D ( B D ikilisi sosuza kadar tekrar eder)

Kısacası A B D düğümleri dışındaki düğümlere asla bakmaz. Bu anlamda tam olmadığını söyleyebiliriz.

BFS (breadth first search , yayılma öncelikli veya sığ öncelikli arama) algoritması ise her zaman için tam kabul edilebilir, sebebi bir sonraki derinlik seviyesine inmeden önce, bulunduğu seviyedeki düğümleri bitirmesi ve bu sayede bütün ağaca bakmayı garanti etmesidir.

**Algoritmanın karmaşıklığına** değinecek olursak. Aşağıdaki şekilde bir formül ile karşılaşırız.

Iddscompleks

Bu formülde görülen terimler şekil kuramında (graph theory) sıkça kullanılan bazı değerlere atıfta bulunur.  
d değerinin derinlik olduğunu daha önce belirtmiştik. b ile gösterilen değer ise dallanma katasyısıdır (branching factor).

Kısaca her düğümün kaç çocuğu olduğu (out order, kaç düğüme gidilebildiği) olarak tutulur. Örneğin tam dolu ikili arama ağacının (binary search tree) b değeri 2′dir. Genelde en kötü durum analizinde bütün çocukların dolu olması ihtimali üzerinde durulur. Ayrıca istatistiksel olarak ortalamam çocuk sayısının da alındığı olur. Örneğin şehirlerin tutulduğu bir şekilde, her şehirden gidilebilecek şehir sayısı değişmektedir. Bu durumda b değeri ortalama değer olarak hesaplanabilir.

Bu açıklama ardından yukarıdaki denkleme tekrar bakarak anlamaya çalışalım.

IDDFS algoritmasında, en altta bulunan düğümlerin 1 kere (d sabitlendiğinde en altta kalan düğümler hangileri ise) dolaşıldığını, d-1 derinliğindeki düğümlerin ise 2 kere dolaşıldığını ve böylece kök düğüme (root) kadar her düğümün, bulunduğu seviyeye göre kaç kere dolaşıldığını hesaplayabileceğimize göre yukarıdaki denklem çıkarılabilir.

Diğer bir deyişle d=0 için kök 1 kere dolaşılırken d=1 olduğunda kök 2 ve kökün çocukları 1 kere dolaşılmış olur. d=3 olduğunda kök 3 kere ve çocukları 2 ve en alttaki çocuklar ise 1 kere dolaşılmış olur. Görüldüğü üzere derinliğin her artışında kök derinlik kadar altındaki her düğüm ise birer azalarak en nihayetinde yapraklar (leaf) tek bir kere dolaşılmış olmaktadır.

Sonuç olarak Yukarıdaki formülün ilk terimi kök (d+1)1 (tek düğüm d+1 kere dolaşılmıştır) , ikinci terimi kökün çocukları (db (kökün çocukları (ki sayısı b’dir) derinlik kadar (ki d ile gösterilir) dolaşılmıştır).

Yukarıdaki bu formülü daha toplu şekilde aşağıda gösterildiği gibi yazabiliriz:  
Iddskompleks2

Algoritmanın dolaşma (arama) zamanı karmaşıklığı yukarıda gösterildiği gibidir. Terimler birleştirildiğinde en kötü durum analizi (worst case analysis) için O(bd) gösterimi kullanılabilir.

Hafıza karmaşıklığı ise O(bd) olarak ifade edilebilir. Bunun sebebi d derinliğindeki bir ağacın her seviyesinde b adet çocuğu bulunması durumunda bd kadar düğüm olacağıdır. Algoritma, çalışması sırasında ilave düğümlere ihtiyaç duymaz ve şeklin kendisi üzerinde dolaşarak sonuca ulaşır.

## **SORU-17: Labirentte yol bulma kodu**

Bu yazının amacı, geri izleme algoritmasının (backtracking algorithm) bir uygulaması olarak, basit bir labirentte yol bulma kodunu JAVA dilinde kodlamaktır. Bu uygulamada herhangi bir yapay zeka yönetmi uygulanmayacaktır. Basitçe kör arama (blind search) yapan ve ihtimalleri sırayla deneyen bir robot uygulaması geliştirilecektir.

Örneğin labirent bilgisinin bir dosyada bulunduğunu ve bizim amacımızın bu dosyayı okuyarak labirentin başlangıcından çıkışına kadar olan yolu bulmak olduğunu düşünelim. Labirent tasarımımızda duvarları 1 ve yolları 0 ile gösterelim. Örneğin aşağıda bir labirent verilmiştir:

1111111  
0010011  
1000011  
1101111  
1100000  
1111111

Yukarıdaki bu labirentte, sol üst köşeyi başlangıç konumu ve sağ alt köşeyi de bitiş konumu olarak kabul edeceğiz. Buna göre labirentin her zaman için ikinci satırının ilk kolonunda başlıyoruz ve labirentin boyutuna göre son satırın bir üstünde en sağ kolonda bitiriyoruz. Amacımız bu iki nokta arasındaki yollardan bir tanesini çözüm olarak bulmak. Ayrıca dosyadan labirent okuyacağımız için, veri yapısı kullanımını kolaylaştırmak amacıyla, labirentin boyutlarını da dosyanın ilk satırında tutalım. Buna göre bir labirent dosyasının içeriği aşağıdaki şekilde olacaktır:

6 7  
1111111  
0010011  
1000011  
1101111  
1100000  
1111111

Dosyanın ilk satırında, labirentin satır ve sütun boyutları verilmiştir.

Algoritmamız, basitçe labirentin başlangıç konumundan yola çıkacak ve yol alternatiflerini değerlendirecektir. Şayet yol tek yönlü ise hareket etmeye devam edecek ancak bir yol ayrımına geldiğinde bu noktayı bir yığında (stack) tutarak çıkmaz bir yola girdiğinde en son aklında tuttuğu bu yola geri dönecek ve farklı bir karar verecektir.

Örneğin yukarıdkai problem için çözümü elle yaparak tasarımımıza başlayalım. Başlangıç durumumuz (1,0) olacak (dizide satır sayısının 0′dan başladığını düşünürsek, ikinci satırın ilk sütunu (1,0) olarak gösterilecektir.

Bitiş durumumuz ise (4,6) olarak belirlenecektir (dizi boyutumuzda azami satır ve sütun değerleri 5,6 olacaktır (boyuttan bir eksik) ve en altın bir üstü satır 4 ve son sütun 6 olacaktır. Dolayısıyla bitiş durumu olarak (4,6) tanımlıyoruz.

(1,0) konumundan, (4,6) konumuna kadar olan çözüm yolunu bulmak için adım adım algoritma tasarımımızı inceleyelim.

Algoritmamız kör arama (blind search) olduğu için, çözüme giden en mantıklı yolu bulmak yerine bütün ihtimalleri deniyeceğiz. Bu ihtimalleri belirli bir sıra ile aramamız gerekiyor. Bu srıalama için algoritmamızı aşağıdaki şekilde tasarlayalım:

*Geldiğim yeri işaretle*

*if(sola gidilmemişse)*

*sola git;*

*else if(sağa gidilmemişse*

*sağa git;*

*else if(yukarı gidilmemişse)*

*yukarı git;*

*else if (aşağı gidilmemişse)*

*aşağı git;*

Yukarıdaki bu tasarımda sol -> sağ -> yukarı -> aşağı sırası tamamen rast gele olarak üretilmiştir ve daha farklı sırada da olabilirdi.

(1,0) konumundan başlayarak yukarıdaki algoritmayı icra ettirelim. İşaretleme için 2 değerini kullanacağız. Öncelikle yığın (stack) değeri boş olarak başlıyoruz (bundan sonra S olarak gösterilecek ve sağdan eklenerek sağdan çıkarma yapılacaktır):

1111111  
0010011  
1000011  
1101111  
1100000  
1111111

S -> boş

İlk değeri işaretliyoruz:

1111111  
2010011  
1000011  
1101111  
1100000  
1111111

(1,0) konumunda tek gidilebilecek komşu, (1,1) konumudur (1,0) konumunun komşuları 0,0 2,0 ve 1,1 olduğuan göre bu değerlere bakıyoruz ve bunlardan 0 olan tek değere gidiyoruz:

1111111  
2210011  
1000011  
1101111  
1100000  
1111111

Benzer şekilde (1,1) konumunun tek 0 olan komşusu (2,1) olan konuma gidiliyor. Benzer şekilde (2,1) konumunun tek komşusu olan (2,2) konumuna gidilerek aşağıdaki durum elde ediliyor:

1111111  
2210011  
1220011  
1101111  
1100000  
1111111

Yukarıdaki son durumda geldiğimiz konum (2,2) ve birden fazla komşumuzda 0 değer var. Yani (2,2) konumunun komşuları olan (2,3) ve (3,2) değerleri 0 olduğu için bu alternatiflerden birisini takip edeceğiz. İşte bu yol ayrımında yığın (stack) kullanmanın sırası geliyor. Bunun sebebi şu anda vereceğimiz kararın doğru olup olmadığını bilmediğimiz ve basitçe hata yapıyorsak buraya geri dönmek istememiz.

S -> (2,2)

Yukarıdaki şekilde karar noktamızı yığına yerleştirdikten (push) sonra algoritmamızdaki öncelikli yol olan sol tarafı deniyoruz. Sol tarafımızda ((2,2)’nin solunda (2,1) vardır) 0 olmadığı için gidemiyoruz. Algoritmamızdaki ikinci ihtimal olan sağ tarafa bakıyoruz (sağın koordinatı (2,3)) ve bu değer 0 olduğu için bu yönde gidiyoruz.

1111111  
2210011  
1222011  
1101111  
1100000  
1111111

Sağa gitme işlemine devam ediyoruz (aşağıya gidemediğimiz için) ayrıca burada bir yol ayrımı bulunduğu için (sağa veya yukarıya gidebiliriz) yığının (stack) içerisine yerleştiriyoruz:

S -> (2,2) , (2,3)

1111111  
2210011  
1222211  
1101111  
1100000  
1111111

Tek alternatif olan yukarıya devam ediyoruz:

1111111  
2210211  
1222211  
1101111  
1100000  
1111111

Tek alternatif olan sola devam ediyoruz:

1111111  
2212211  
1222211  
1101111  
1100000  
1111111

Netice itibariyle alternatifi kalmayan (hiçbir yönde 0 olmayan) bir duvar ile karşılaştık ve bu noktadan sonra çıkma ihtimalimiz kalmadığı için yığındaki en üstte bulunan ve en son karar verdiğimiz noktaya dönüyoruz.

Bu nokta (2,3) koordinatlarındadır ve pop edildikten sonra stack aşağıdaki şekildedir:

S – > (2,2)

Pop ettiğimiz (2,3) noktasında da gidilecek hiçbir alternatif bulunmuyor. Mecburen yığındaki (stack) bir sonraki noktaya dönüyoruz:

S -> Boş

Döndüğümüz koordinat (2,3) noktası ve buradaki tek alternatif aşağı yönde bulunan 0 değeri:

1111111  
2212211  
1222211  
1121111  
1100000  
1111111

Bu aşamadan sonra, hiçbir alternatif kalmadan 0 olan yollar izlenerek sonuca ulaşan yol bulunmuş oluyor.

**Kodlama**

Yukarıdaki yaklaşımın JAVA dilinde kodlanmış hali yazının sonunda verilmiştir. Bu kodun kısaca açıklamasını bu bölümde anlatacağım. Kodumuzda temel olarak iki sınıf bulunuyor:

* currentInfo
* Maze

currentInfo sınıfı, anlık olarak matrisin hangi konumunda bulunduğumuzu tutmaya yarıyor. Çok basit şekilde satır ve sütün koordinatlarımızı barındırıyor. Ayrıca bu değerlere erişmeyi sağlayan inşa sınıfları (constructors) ve kapsülleme fonksiyonları (encapsulation) bulunuyor.

Maze sınıfı ise geri kalan herşeyi yapan ana sınıfımız. Bu sınıfta getMaze() isimli fonksiyon, dosyadan labirent bilgilerini okumak için kullanılıyor. solveMaze fonksiyonu da labirentteki sonucu bulmaya yarayan yukarıdaki işlemleri çalıştırıyor.

Yukarıdaki anlatımda geçen ve S harfi ile ifade ettiğimiz yığın (stack) ise kodumuzda path ismi ile tanımlanmış java.util.Stack sınıfından tanımlanmış bir yığın. Buradaki yığın, karar verirken atılan her adımı tutmakta. Yani yukarıdaki labirent çözümünde geçen ve 2 olarak işaretlerken, birden fazla 0 ihtimali bulunan dönüm noktalarını tutuyor.

Yığının en temel 3 fonksiyonu ise aşağıdaki şekildedir:

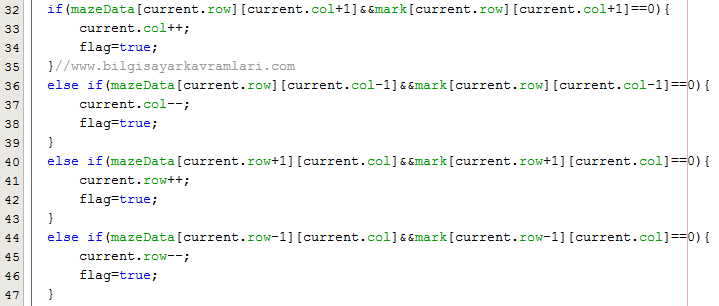
* Pop() yığında bulunan en üstteki değeri alarak bir değişkene döndürür.
* Push() yığının en üstüne, parametre olarak aldığı veriyi yerleştirir.
* Peek() yığının en üstündeki veriyi alır ancak pop fonksiyonunun aksine bu veriyi silmez. Bu fonksiyon bazı kaynaklarda Top() olarak da geçmektedir ancak java.util.Stack sınıfında peek ismi ile tanımlanmıştır.

Kodumuzda kullandığımız iki temel dizi ise aşağıda verilmiştir:

* mazeData[][] iki boyutlu bu dizi, dosyadan okunan ve hafızada labirent bilgisini tutan dizidir.
* Mark[][] diğer iki boyutlu dizimiz ise sonuca giden yolu tutar.

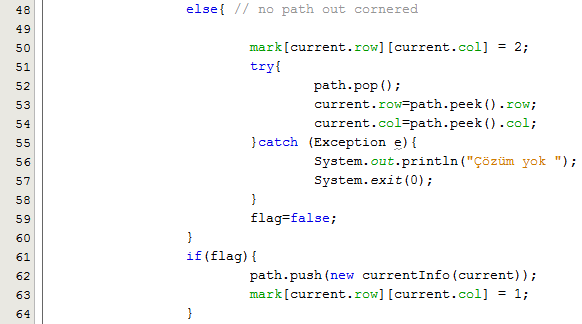
İkinci diziye neden ihtiyaç duyduğumuz sorusunu şu şekilde cevaplayalım. Dizimizi dolaşırken, yanlış yollara sapabiliyoruz, geçtiğimiz her yeri basarsak bu sonuçta görmek istediğimiz çözüm yolu olmayacaktır. Ayrıca labirentte dolaşırken, çözüme giden yolu 2 ile işaretliyoruz ama hatalı yolları da 2 ile işaretliyoruz. Dolayısıyla labirenti dolaştıktan sonra 2 ile işaretli yolları basarsak bu yollarda da sonuca gitmeyen değerlerin basılması söz konusu. Çözüm olarak ayrı bir dizide, sonuca giden yoldaki koordinatları ayrıca tutuyoruz.

Koddaki bazı satırları anlatmamız gerekirse:



Şekil 1: Kod

Yukarıdaki kod parçasında, bulunduğumuz konumdan (current) sonraki gidebileceğimiz yönleri sırasıyla deniyoruz. İlk if koşulunda mazeData isimli dizide (yani labirentin verilerinin tutulduğu dizi) bulunan ve mevcut konumun bir sağında yer alan (col + 1 işleminin anlamı bir sonraki kolon demektir) değere gidilmesi ve bu değerin daha önce mark dizisinde işaretlenmemiş olması (gidilmemiş olması ) durumu kontrol edilmektedir. Ardından sırasıyla, sol, aşağı ve yukarı yönler için aynı kontroller yapılmaktadır. Yazıda anlatılan sıradan bu anlamda farklıdır.

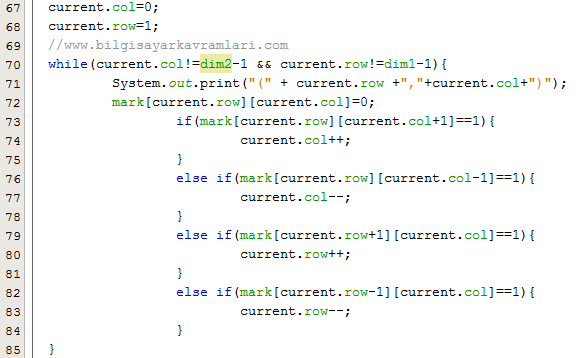


Şekil 2: Kod

Yukarıdaki kod parçasında ise, bir önceki kod parçasının devamı niteliğinde, şayet 4 yöndede hareket edilemiyorsa, çıkmaz bir yola girilmiş neticesine varılabilir. Bu durumda mevcut durum dolaşıldı anlamında, 2 ile işaretlendikten sonra, yığından (stack) pop fonksiyonu ile dönülecek en son karar anına bakılıyor.

Şayet yığın bir şekilde boş olursa (ki bu durum try/catch blokları arasında istisna kabz edilerek (exception handling) çözülmüştür) bu durumda labirentin bir çözümü olmadığını söyleyebiliriz çünkü hem bir çıkmaz yol bulduk hem de geriye döneceğimiz hiçbir yol ayrımı bulunmuyor demektir.

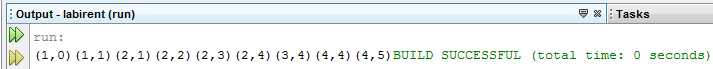
Netice itibariyle, 61. Satıra kadar ulaştıysak, bu satırda flag değişkeni true değerinde olup, yukarıdaki 4 iften birisine girerek bir yöne hareket etmiş olabilir. Bu durumda gidilen bu yok, mark dizisinde işaretlerenerek yığına yerleştiriliyor. Ayrıca flag değişkenin false olduğu tek ihtimal olan çıkmaz yol durumunda da if bloğu çalışmıyor.



Şekil 3: Kod

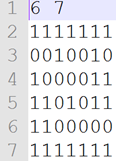
Yukarıda alıntılıanan kod parçası ise bütün işlemler bitip, labirent bir sonuca ulaşacak şekilde dolaşıldıktan sonra, mark dizisi üzerinden dolaşılan yolu ekrana basmaktadır. Döngü, labirent boyutları olan dim1 ve dim2 sınırlarında dönmekte, her adımdaki konum 71. Satırda ekrana basılmakta ve sonraki konum da 4 farklı koşul konrolü ile işaretlenmiş olan yöne doğru değiştirilmektedir.

Kodun çalışması sonucunda, örnek ekran çıktısı aşağıda verilmiştir:



Şekil 4: Çıktı

Görüldüğü üzere labirentteki bütün yolları dolaşarak, sonuca ulaşan yolları ekrana basmıştır(lütfen basma işlemi sırasında, 71. Satırda olduğu üzere önce satır sonra sütun koordinatının basıldığına dikkat ediniz). Bu örnek çalışma, aşağıdaki labirent içindir:



Şekil 5: Çıktı

Ayrıca farklı bir dosya içeriği, projenin bulunduğu dizindeki maze.dat dosyasının içerisine yerleştirilerek çalıştırılabilir.

## **SORU-18: Işın Araması (Beam Search) hakkında bilgi veriniz.**

Bu yazının amacı, özellikle yapay zeka çalışmaları altında yer alan sezgisel arama algoritmalarının (heuristic search algorithms) bir çeşidi olan ışın araması konusunu anlatmaktır.  
Işın arması “en iyi ilk arama” (best first search) tipi arama algoritmalarındandır ve amacı hafıza ihtiyacını azaltmaktır.

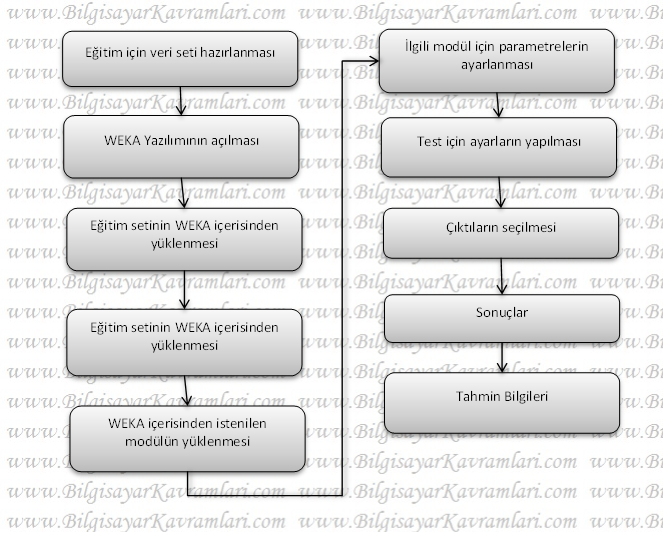
En iyi ilk arama yaklaşımını kullanan arama algoritmaları, arama yaptıkları şekil (graph) üzerinde bütün çözüm ihtimallerini belirli bir sezgisel fonksiyona göre sıralarlar. Diğer bir deyişle arama işlemi sırasında sezgisel bir fonksiyona göre, aramada bazı kararlara öncelik verirler. Bu sezgisel fonksiyonun amacı, karar anında verilen kararın, sonuç durumuna (goal state) ne kadar yakın olduğunu yargılayabilmektir.   
Işın aramasının bir özelliği de, daha önceden belirlenmiş ve sınırlı sayıdaki, yapmış olduğu seçimleri saklamasıdır.   
Işın araması, bir arama ağacı (search tree) oluştururken, sığ öncelikli arama (breadth first search) yaklaşımı kullanır. Buna göre ağacın herhangi bir seviyesinde, gidilebilecek düğümlerin sezgisel değerlerini hesaplar ve bunlardan sadece belirli sayıdaki düğümü, daha sonra geri dönülmek üzere hafızasında tutar. Bu değere ışın aramasına özgü olarak ışın genişliği (beam width) ismi verilir.   
Işın genişliği arttıkça daha fazla düğüme bakılmaktadır. Işın genişliği azaldıkça da daha fazla düğüm budanmaktadır (prunning). Budama işleminin dez avantajı, şanssız bir şekilde en iyi sonucu verecek düğüme giden yolu budama riskidir. Ancak bu riskten dolayı bütün düğümleri tutmak (yani hiçbir düğümü budamamak) gibi bir karar verirsek, bu karar sonucunda da çok daha fazla hafıza ve işlem ihtiyacımız olacaktır. İşte ışın araması, bu iki karar arasında en iyi (optimum) noktayı bulmayı amaçlar.   
Bu amaca yönelik olarak, ışın araması sabit ışın genişliği (fixed beam width) veya değişken ışın genişliği (variable beam width) kullanacak şekilde kodlanabilir. Değişken ışın genişliği alan arama algoritmasında, öncelikle ışın genişliği asgari değerde tutularak arama yapılır. Aranan en iyi sonuca ulaşılamazsa, ışın genişliği arttırılarak arama işlemi sürdürülür. Beklenen sonuca ulaşılana kadar da ışın genişliği arttırılır.

Işın aramasının kullanıldığı yerlerden birisi de bilgisayarlar tarafından otomatik olarak yapılan tercüme işlemleridir. Bilindiği üzere bir kelime, bilgisayar tarafından işlendiğinde, birden fazla anlama gelebilmektedir. İnsanlar kelimenin kullanımından (syntax, pragma) kelimenin anlamını kısa sürede doğru olarak anlayabilmekte ancak bilgisayarlar için bu anlama süreci ve doğru anlamın bulunması (semantics), hatırı sayılır bir süre almaktadır. Çözüm olarak ışın araması algoritması, çok sayıdaki çeviri ihtimallerinden sadece belirli bir kısmını işleyerek doğru tercümeyi yapmak için kullanılabilir.

## **SORU-19: WEKA ile SVM hakkında bilgi veriniz.**

Bu yazının amacı, site üzerinde gelen bir soru üzerine, WEKA programı üzerinde, destekçi vektör makinesinin (SVM) gerçekleştirilmesi ile ilgili bir uygulama anlatmaktır.

WEKA üzerinde genel olarka bir işlem yapmak için aşağıdaki adımlar izlenebilir:

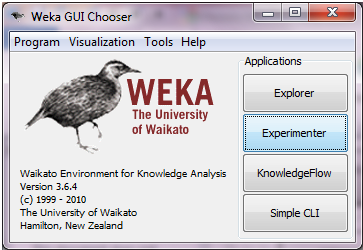


Şekil 1: Akış şeması

Yukarıdaki adımlara göre, öncelikle, işlenmek için hazır bir veri kümemiz bulunması gerekiyor.

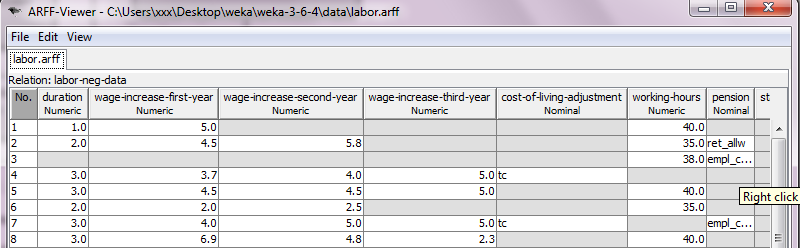
Yazımızdaki amacımız, konuyu öğrenmek olduğu için, WEKA kurulumu ile gelen örnek veri kümelerinden birsini, bu yazı kapsamında kullanacağız.

1. **Adım:** WEKA’yı açıyoruz:



Şekil 2: WEKA

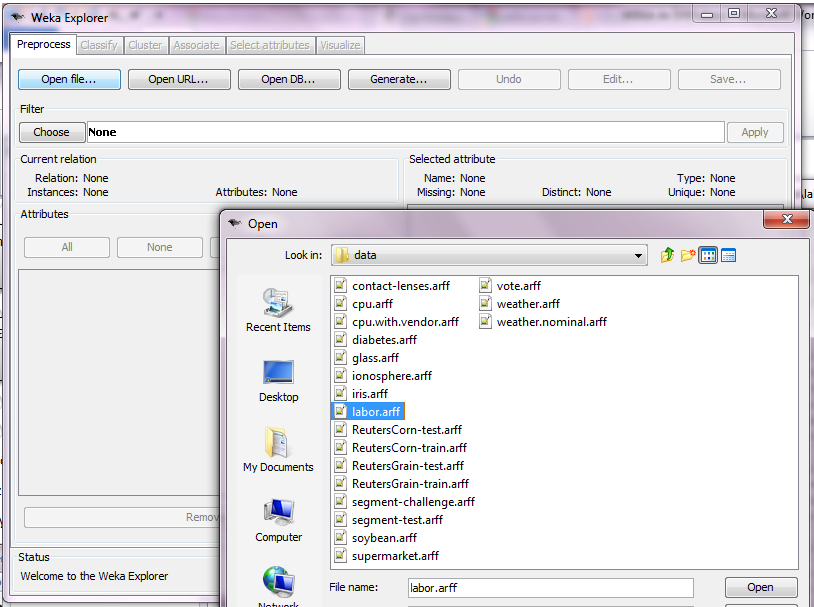
**2. Adım:** Önce veri kümemize bir göz atalım. WEKA’nın kurulumu ile birlikte gelen data dizini altındaki, labor.arff dosyasını görmek için, açılan ilk weka ekranında (yukarıdaki resimde temsil edilen), “Experimenter” düğmesine basıyoruz. Açılan ekranda, File / Open seçeneklerini menülerden seçtikten sonra, dosya seçicisinden “labor.arff” dosyasını işaretleyip açıyoruz:



Şekil 3: Dosya açma

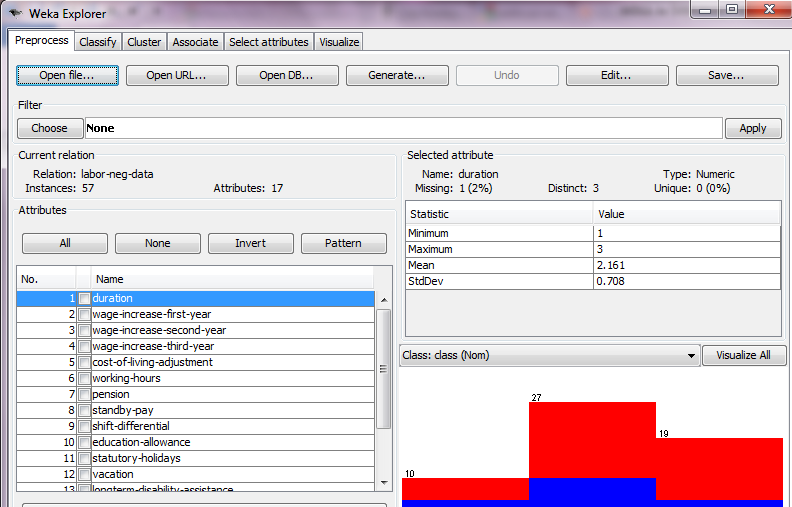
Kullanacağımız örnek veri kümesi yukarıdaki şekildedir. Hızlı bir bakışla, verilerin, çalışan verileri olduğunu ve her çalışanın maaş, çalışma süreleri, sosyal güvenliğine ait bilgilerin yer aldığı bir veri kümesi olduğunu söyleyebiliriz. Şimdiki adımımız bu veri kümesi üzerinden SVM algoritmasını çalıştırmak olacak**.**

**3. Adımda** WEKA’nın ilk açılan ana ekranına dönerek, bu sefer “Explorer” düğmesine basıyoruz ve WEKA içerisinde yer alan fonksiyonları kullanacağımız ekran açılıyor.



Şekil 4: Open

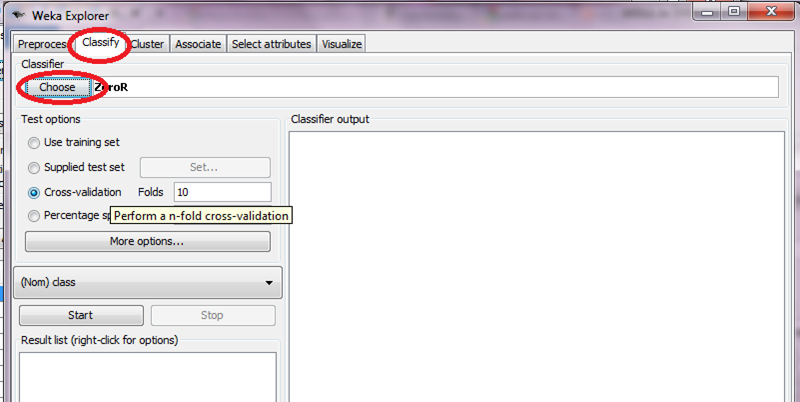
Yukarıdaki açılan Weka Explorer ekranında bulunan ilk sekme olan Preprocess sekmesinde (ki ilk açıldığında bu sekme açık gelir), bulunan “Open File” düğmesine tıklayarak dosya seçicimizi açıyoruz ve yine Weka’nın kurulumuyla gelen data dizini altında bulunan “labor.arff” dosyasını seçerek açıyoruz.



Şekil 4: Ön işleme sonuçları

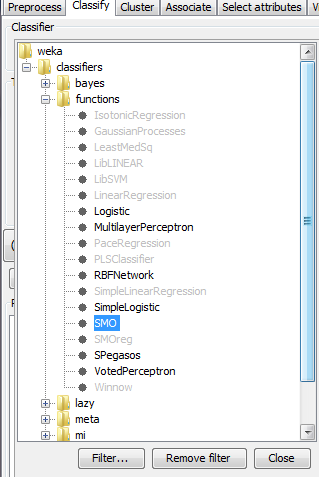
Ekranda, seçilen bu dosyanın ön işleme sonuçları görülüyor. Buna göre sol taraftaki listede, ilk başta görüntülediğimiz arff dosyasının her bir kolonu yer almakta olup, her kolona tıklandığında, bu kolon ile ilgili veri detayları sağ tarafta çubuk grafik olarak görülmektedir.

**4. Adımda** WEKA’nın sınıflandırma işlemi için ilgili modülün yüklenmesine geçecegiz. Bu işlem için, ekranın üzerinde bulunan “Classify” sekmesine geçiyoruz.



Şekil 5: Seçme işlemi

Sekmeye geçtikten sonra “Choose” düğmesine tıklayarak kullanacağımız modülü seçeceğiz. Burada WEKA içerisinde yüklü olan ve konu ile ilgisi olan 3 ayrı modülden bahsetmekte yarar vardır:



Şekil 6: Fonksiyon elemanları

Listede görülen classifiers klasörü altındaki functions elemanlarıdır. Bunların tamamı, sınıflandırma için kullanılan fonksiyonlardır. Tam bu noktada 3 ayrı alternatiften bahsedelim:

**SMO**

Yani birden fazla kümeye, veriyi bölmeye yarar. Listede bulunan SMO, SVM benzeri bir yapıdadır.

Sequential Minimal Optimisation kelimelerinin baş harflerinden oluşan SMO, esas itibariyle Suppor Vector kullanan bir algoritmadır. Algoritmanın detayı için WEKA dokümantasyonunda da yer alan aşağıdaki makalelere bakılabilir:

* J. Platt: Fast Training of Support Vector Machines using Sequential Minimal Optimization. In B. Schoelkopf and C. Burges and A. Smola, editors, Advances in Kernel Methods – Support Vector Learning, 1998.
* S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, K.R.K. Murthy (2001). Improvements to Platt’s SMO Algorithm for SVM Classifier Design. Neural Computation. 13(3):637-649.
* Trevor Hastie, Robert Tibshirani: Classification by Pairwise Coupling. In: Advances in Neural Information Processing Systems, 1998.

**LibSVM**

SVM sınıflandırıcısının en önemli kütüphanesi, bir doktora tezi sırasında ortaya çıkmış olan ve libSVM olarak bilinen açık kaynak kodlu kütüphanedir. Bu kütüphane, ne yazık ki WEKA kurulumu ile gelememektedir. Ancak bu kütüphaneyi WEKA altında kullanmanın iki yolu bulunur. İlki, kütüphaneyi WEKA’ya tanıtmaktır. Bunun için, yukarıdaki temsili resmi verilen listede bulunan libSVM kullanılabilir. Öncelikle kütüphane, aşağıdaki adresten indirlmelidir:

[http://www.csie.ntu.edu.tw/~cjlin/libsvm/](http://www.csie.ntu.edu.tw/%7Ecjlin/libsvm/)

Ardından weka’nın paket yöneticisi tarafından kurulmalıdır. Wekanın paket yöneticisi ise WEKA’nın classpath tanımının yapıldığı bir konsolda aşağıdaki komutu vererek çalıştırılabilir:

java weka.core.WekaPackageManager

Diğer bir yöntem ise, indirilen libsvm kütüphanesinin WEKA classpath’ine eklenmesidir. İlgili ayarlar yapıldıktan sonra 4. adımda bahsettiğimiz sınıflandırmalardan birisi olarak libSVM kullanılabilir.

LibSVM kullanımı için ikinci bir yöntem ise, LibSVM ve WEKA programlarını, aynı anda içeren wlsvm kütüphanesini kullanmaktır. Bu kütüphaneye de aşağıdaki adresten erişilebilir:

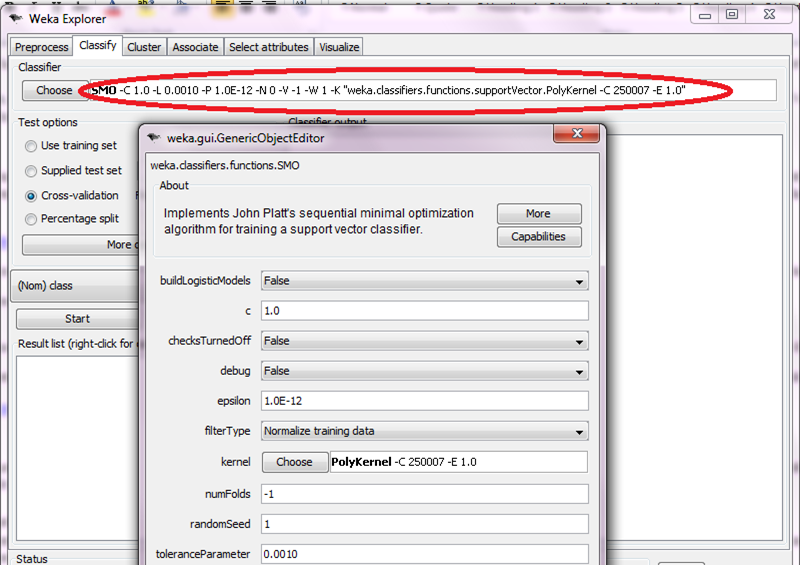
http://www.cs.iastate.edu/~yasser/wlsvm/

**SMOreg**

SVM ile ilgili, weka içerisinde bulunan 3. bir seçenek de SMOreg seçeneğidir. Bu seçenek, SVM üzerinde Regression (ilkelleme, regrezizasyon) uygulamaya yarayan kütüphanedir. Bu kütüphane için de detaylı bilgi, aşağıdaki kaynaklardan edinilebilir:

* Alex J. Smola, Bernhard Scholkopf (1998). *A Tutorial on Support Vector Regression*. NeuroCOLT2 Technical Report Series – NC2-TR-1998-030.
* S.K. Shevade, S.S. Keerthi, C. Bhattacharyya, K.R.K. Murthy, *Improvements to SMO Algorithm for SVM Regression*. Technical Report CD-99-16, Control Division Dept of Mechanical and Production Engineering, National University of Singapore.

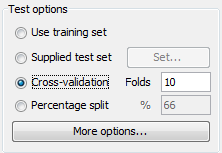
**5. Adımda**, Yukarıdaki seçeneklerden birisi seçildikten sonra, seçilen modüle göre ayarlama yapılması gerekemektedir. Örneğin SMO kütüphanesi üzerinden ilerleyecek olalalım. Bu modülün parametre ayarlarını, Object Editor ile yapabiliriz. ObjectEditor’ü açmak için ekranda bulunan ve fonksiyonun parametrelerini içeren metin kutusuna tıklamak yeterlidir:



Şekil 7: Fonksiyon parametreleri

Açılan bu ekranda, ilgili parametre ayarları yapıldıktan sonra, sınıflandırma fonksiyonunun çalışması için, WEKA Explorer ekranında bulunan “Start” düğmesine basmak yeterli olacaktır. Sonuçlar ve sonuçların değerlendirmeleri (hata miktarları gibi), sonuç ekranında görülecektir. Buradaki sonucu etkileyen diğer bir ayar da, kullanım sırasında eğitim ve test için izlenecek stratejidir.

WEKA doğrudan, 4 farklı özellik desteklemektedir. Bunlar sırasıyla aşağıda anlatılmıştır:



Şekil 8: WEKA özellikleri

Trainin set seçeneği, seçilen verinin sadece eğitim amaçlı kullanılmasını sağlar. Bu seçenekte önce eğitim yapılıp ardından ikinci bir küme, test amaçlı kullanılabilir. Bu seçenek de yine yukarıdaki ekranda bulunan “Supplied test set” seçeneği ile gösterilen küme üzerinde çalışır.

“Cross-validation” seçeneği, veri kümesine belirtilen (folds sayısı olarak belirtilen) sayıda kümeye böler. Alt kümelerden birisini eğitim kümesi olarak kabul ederek sistemi eğitir. Ardından bu eğitim sonucunu diğer bir alt küme üzerinde sınar (ki bu kümeye sınama kümesi (validating set) veya test kümesi (test set) ismi verilir). Bu işlemi belirtilen küme sayısı kadar tekrarlayarak sisemi iyileştirmeye çalışır.

Son seçenek olan “percentage split” ise, verilen oranda kümeyi ikiye böler ve ilk kümeyi eğitim ikinci kümeyi ise test amaçlı kullanır. Örneğin %66 bölümün anlamı, kümenin ilk %66 kısmının eğitim sonraki %34 kısmının ise test amaçlı kullanılacağıdır.