DATA MINING NOTES BY EMEL KAYACI & ANIL TAYSİ

İÇİNDEKİLER

1. Clustering 1
1.1 Özellikleri 1
1.2 Verilerin benzerliği nasıl belirlenir? 1
1.2.1 Aralık değerli veri 1
1.2.2 Binary veri 2
1.2.3 Nominal veri 2
1.2.4 Ordinal veri 2
1.2.5 Oransal veri 2
1.2.6 Sonuç 2
1.3 Kümeler arası ölçütler (Inter Cluster Distance) 3
1.4 Cluster kalitesini ölçen metrikler 3
1.5 Bazı önemli clustering çeşitleri 3
1.5.1 Agglomerative 3
1.5.2 K-Means 4
1.5.3 K-Medoid 5
1.6 Konveks olmayan kümeler k-Mediod clustering ile ifade edilemezler. 5
2. Decision Trees 6
2.1 Classification vs. Prediction 6
2.2 Inductive Learning 6
2.2.1 Find -S Algorithm 7
2.2.2 List-Then-Eliminate Algorithm 8
2.2.3 Candidate Elimination Algorithm 8
3. Decision Trees 9
3.1 Information Gain 10
3.2 ID3 algoritmasının biasi: Küçük ağaçları her zaman büyük ağaçlara tercih etmektir. 10
3.3 Occam's Razor 11
3.4 Karar Ağaçlarında Overfitting 11
3.5 C4.5 Gain Ratio 11
3.6 Gini Index 12
3.7 ID3 algoritmasının hipotez uzayı nedir? 12
4. Neural Networks 12
4.1 Stochastic vs. Gradient 13

4.2 Multilayer Networks 13
4.3 Local Minima için Çözümler 14
4.4 Penalty term 14
4.5 Cross entropy 14
4.6 Conjugate gradient 15
5. EKLER 15
5.1 Bayesian Learning 15
5.2 Gibbs Algoritması 16
5.3 Öğrenme Çeşitleri 16

1. Clustering

1.1 Özellikleri

1. Farklı tiplerdeki değişkenler için kullanılabilir.
2. Dinamik veri ile iyi başa çıkabilmektedir. Yeni veri gelince direkt olarak hangi cluster'a ait olduğunu bulabilmektedir.
3. Değişik şekillerde cluster'lar bulabilmektedir. Yalnızca convex olmayan şekillerde bulamamaktadır.
4. Domain bilgisine ihtiyaç olmadan benzer özellikleri kendisi bulup gruplandırma yapabilmektedir.
5. Yorumlama açısından kolaydır.
6. Girdilerin sırasının herhangi bir önemi bulunmamaktadır.
7. High dimensional olabilir.

İyi bir clustering 2 temel yapıyı içerir.

1. **High intra-class similarity** (Aynı cluster içindeki elemanlar arası benzerlik oldukça fazladır.)
2. **Low inter-class similarity** (Farklı cluster içindeki elemanlar arası benzerlik oldukça azdır.)

1.2 Verilerin benzerliği nasıl belirlenir?

Uzaklık fonksiyonları veri tipine göre farklılık gösterir.

1.2.1 Aralık değerli veri

Veri öncelikle normalleştirme işleminden geçirilir. Bunun da iki yolu bulunur.

1. Ortalamadan yaralanmak.

$$s_f = \frac{1}{n}(|x_{1f} - m_f| + |x_{2f} - m_f| + ... + |x_{nf} - m_f|)$$

2. Normal dağılımdan (z skoru) yararlanmak.

$$z_{if} = \frac{x_{if} - m_f}{s_f}$$

Normalleştirildikten sonra en yaygın kullanılan teknik *Minkowski* uzaklığıdır.

$$d(i,j) = \sqrt[q]{(|x_{i_1} - x_{j_1}|^q + |x_{i_2} - x_{j_2}|^q + ... + |x_{i_p} - x_{j_p}|^q)}$$

q = 1 ise Manhattan, q = 2 ise Euclidean uzaklığı olur.

$$d(i,j) = |x_{i_1} - x_{j_1}| + |x_{i_2} - x_{j_2}| + ... + |x_{i_p} - x_{j_p}| \qquad d(i,j) = \sqrt{(|x_{i_1} - x_{j_1}|^2 + |x_{i_2} - x_{j_2}|^2 + ... + |x_{i_p} - x_{j_p}|^2)}$$

q arttıkça aradaki farklılıkları büyütür ve daha fazla netleştirir. Farklı sütunlarda farklı üstel değerleri kullanabiliriz.

1.2.2 Binary veri

Özellikle resim verisi (piksel) saklamakta kullanılır.

Contingency tablosu ile ifade edilir.

		Object *j*		
		1	0	*sum*
Object *i*	1	*a*	*b*	*a+b*
	0	*c*	*d*	*c+d*
	sum	*a+c*	*b+d*	*p*

i=1 ve j=1 ise benzer özelliğin resimde bulunduğunu i=1, j=0 veya i=0, j=1 özelliğin resimde bulunmadığı ifade ederken i=0, j=0 hiçliği ifade eder bu nedenle d bazen alınmaz.

$$d(i,j) = \frac{b+c}{a+b+c+d}$$

$$d(i,j) = \frac{b+c}{a+b+c}$$

Benzerlik metriğine *Jaccard* denir ve pay kısmında a bulunur.

1.2.3 Nominal veri

Binary değişkenlere benzer fakat ikiden fazla durum içerebilir. Örneğin mavi, kırmızı, sarı, yeşil renkler.

m = # of mathces p = total # of variables

$$d(i,j) = \frac{p-m}{p}$$

1.2.4 Ordinal veri

Order önemlidir.

3 adım ile hesaplanır. İlk 2 adım dönüşümü içerir daha sonra 1.2.1'de bahsedilen aralıklı değerler için kullanılan yöntemler kullanılır.

1. x_{if} yerine rank değeri yazılır.
2. Aşağıdaki formül kullanılarak her bir rank 0-1 aralığına sıkıştırılır.

$$z_{if} = \frac{r_{if} - 1}{M_f - 1}$$

1.2.5 Oransal veri

Logaritmik dönüşümler kullanılılarak ordinal veri şeklinde işleme devam edilir.

1.2.6 Sonuç

$$d(i,j) = \frac{\sum_{f=1}^{p} \delta_{ij}^{(f)} d_{ij}^{(f)}}{\sum_{f=1}^{p} \delta_{ij}^{(f)}}$$

1.3 Kümeler arası ölçütler (Inter Cluster Distance)

1. **Single link:** İki küme arasındaki en kısa mesafeye bakar. O(n)
2. **Complete link:** İki küme arasındaki en uzun mesafeye bakar. O(n)
3. **Average:** Her bir kümedeki elemanlar arası ortalama uzaklıklara bakar. En doğru sonuç verir fakat en maliyetlisi de budur. $O(n^2)$
4. **Centroid:** Ağırlık merkezine bakar. Gerçek nesneye ait olmayabilir, hayali bir nokta. Maliyetsizdir çünkü toplamı tutuyorsak yeni veri eklendiğinde tekrar tüm değerlere bakmaya gerek yoktur. Elemanların geliş sırasından eklenir. O(1)
5. **Medoid:** Gerçek bir değeri ifade eder. Baştan tüm değerleri gezmesi lazım. O(n)

1.4 Cluster kalitesini ölçen metrikler

Bunlar stopping condition olabilirler. Yani kalite belli bir seviyeye geldiğinde kümeleme işlemine son verilip var olan kümeler sonuç olarak gösterilebilir.

1. **Centroid:** "Middle" of the cluster. Ağırlık merkezi bulunurken n tane elemanın toplamlarının ortalamasının alınması gerekmektedir. Bütün elemanlar bir kez gezildiğinden O(n) olur. Yukarıda O(1) ile belirtilmiş centroid yeni bir veri eklendiğinde güncelleme zamanını göstermektedir.
2. **Radius:** Her bir elemanın centroid'e olan uzaklıklarının ortalamasını verir. (Yarı çap) Değerin küçük olması hedeflenir çünkü küçükse elemanların küme içinde çok yayılmadığı sonucuna varabiliriz. Bunun için önce centroid hesaplanmalıdır. Ayrıca her bir elemanın centroid'e olan uzaklıkları da hesaplanmalıdır. Her iki işlem de O(n) sürede gerçekleşir. O(n) + O(n) sonucunda zaman karmaşıklığı O(n) olarak elde edilir.
3. **Diameter:** Her bir elemanın her birine olan uzaklıklarının ortalamasını verir. Değer küçük ise küme içinde yoğunluk yüksektir demektir. Her bir üye diğerlerine ne kadar yakın/uzak. N adet nokta için n-1 kere hesaplama işlemi yapıyoruz. O(n*(n-1)) $O(n^2)$

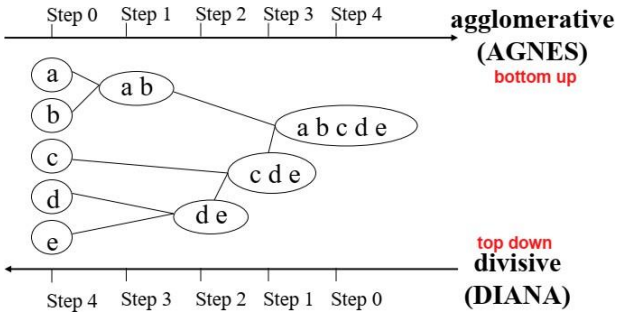
1.5 Bazı önemli clustering çeşitleri

1.5.1 Agglomerative

Her adımda bir alt küme oluşturulur. Kümeler birleşerek en sonunda tek bir küme elde edilir. Her bir adımda toplam eleman sayısımız da bir azalmaktadır. Böylece zaman karmaşıklığı,

n*(n-1) + (n-1)*(n-2) + (n-2)*(n-3) + … + 3*2 + 2*1'den $O(n^2)$ olur.

Fakat her adımı tutmamız gerektiğinden space complexity $O(n^3)$'tür. Bu nedenle sık kullanılmaz.



1.5.2 K-Means

1. Kullanıcıdan kaç adet cluster oluşturulacağına dair girdi istenir. Bu bir olumsuz özellik olup farklı k değerlerinin denenip cluster içi kalite metriklerinden seçim yapmamız gerekir. Fakat k değerleri küçük seçildiğinden işlem gücü açısından çok sorun olmaz.

2. Seçilmiş k adet nokta random olarak belirlenir.

3. Her bir veri hangi noktaya daha yakın olduğunu bulur. Voronoi diyagramı ile ayrılmış düzlemler bulunur. O(n*k)

4. Kümelenmiş noktalar kendi centroid'lerini hesaplar. Böylece 2. adımda belirlenmiş random noktalar taşınmış olur. O(n)

5. Bu işlem aşağıdaki koşullardan herhangi biri sağlanana kadar devam eder.

t*((n*k) + n))

1. Centroid move hamleleri kalmadığı zaman,

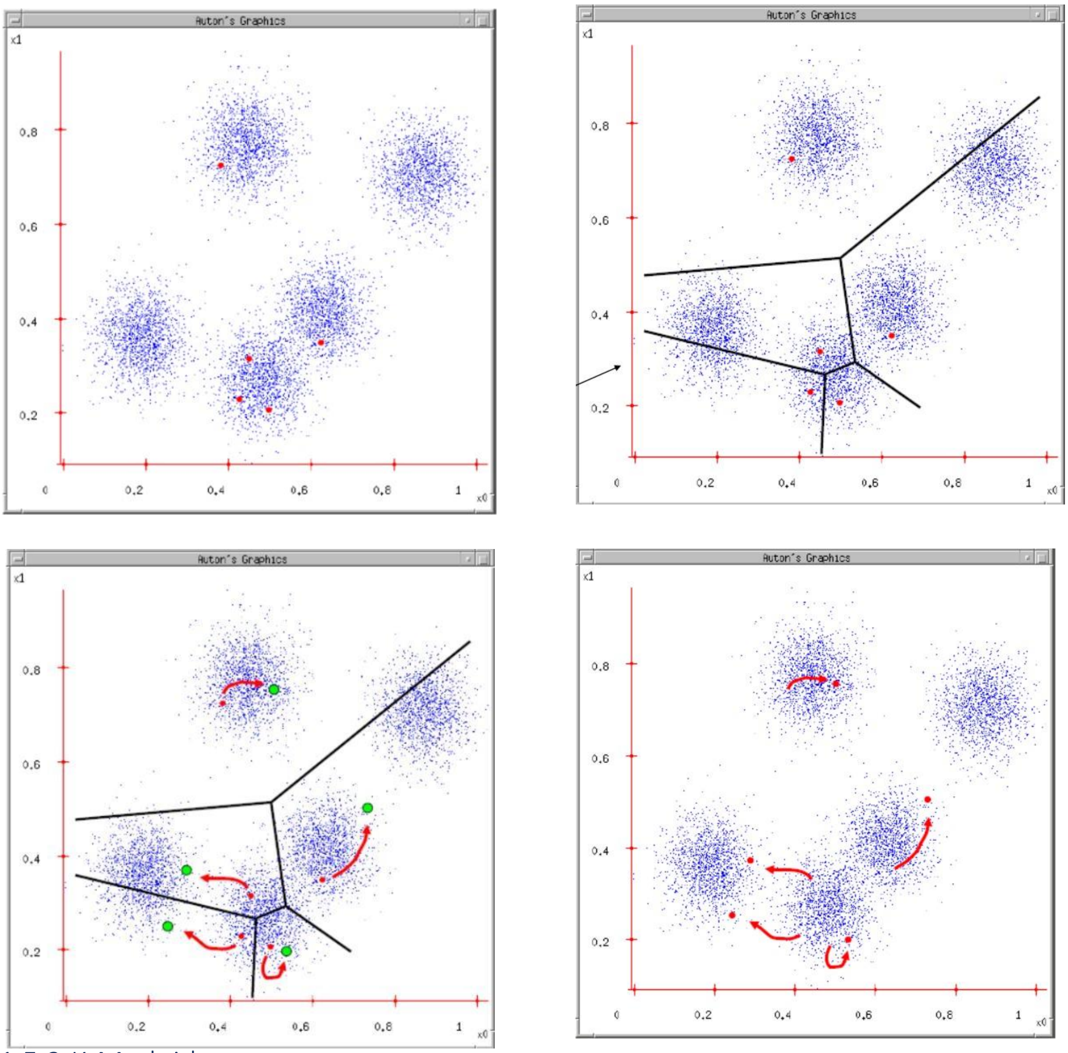
2. Belli sayıda iterasyon sonucunda,

3. Cluster kalitesi belli bir seviyeye ulaştığında

K-Means zaman karmaşıklığı yorumu

n kayıt, t iterasyon, k cluster için O(t*k*n) = O(n) k << t << n

n, k ve t'ye göre çok büyük olduğundan O(n) kabul edilir.

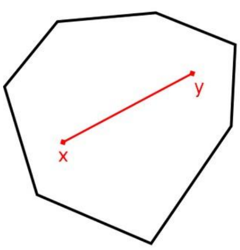


1.5.3 K-Medoid

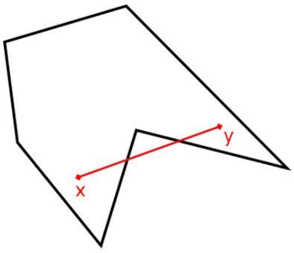
Merkez için centroid yerine medoid kullanılmasıdır. K-means'in aksine gürültüden etkilenmez. $O(t*k*(n-k)^2)$

Her cluster bir üye alır ve kalanı tek bir cluster içerisinde toplanırsa toplam üye n-k olur, medoid hesaplanırken $O(n^2)$ olduğundan worst case $(n-k)^2$ olur.

1.6 Konveks olmayan kümeler k-Mediod clustering ile ifade edilemezler.



Convex region



Non-convex region

In Euclidean space, an object is convex if for every pair of points within the object, every point on the straight line segment that joins them is also within the object.

Konveks olmayan şekillerde ağırlık merkezi küme sınırları içinde olmayabilir. Bu durumda küme uzaklıkları belirlenirken 1.3 kısmında belirtilen bu sınırın dışındaki noktaya en yakın fakat şekil üzerindeki nokta baz alınarak hesaplamalar yapılır. Fakat bu noktanın küme sınırı olması iyi sonuçlar üretmeyecektir.

Hierarchical clustering'de konveks olmayan şekiller sorun yaratmaz.

2. Decision Trees

2.1 Classification vs. Prediction

- Classification ile bir sınıftaki veriyi tahmin ederiz, prediction'da ise bir sayısal değeri tahmin ederiz.
- Prediction daha net bir sonuç olduğundan daha risklidir bu nedenle daha çok classification kullanılır.
- Her prediction problemi belli bir kesinlik kaybederek classification problemine dönüştürülebilir.

2.2 Inductive Learning

Induction: Model üretmek, train, knowledge extraction

Deduction: Knowledge kullanarak yeni label oluşturmak

Inductive learning yes, no gibi iki değerleri (binary) classification problemlerini çözmekte kullanılır.

Hipotezlerin gösterimi

Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

?: Her değer olabilir

'Warm': Yalnızca tek değer olabilir. Örnekte Warm verilmiş.

0: Hiçbir değer olamaz.

Hatırlatma: İki ana bias çeşidimiz bulunur. Bunlar restriction (inductive) ve search (preference) bias. Baştan beri alternatifler hiç dahil edilmezse restriction, model oluşturulurken elemeler yapılıyorsa search bias olur.

Aldo sadece soğuk havalarda nem oranı yüksekken spor yapmayı sever hipotezi şu şekilde ifade edilir:

<?, Cold, High, ?, ?, ?>

En **genel** hipotez her durumda spor yapmasıdır.

<?, ?, ?, ?, ?, ?>

En **spesifik** hipotez hiçbir durumda spor yapmamasıdır.

<0, 0, 0, 0, 0, 0>

2.2.1 Find -S Algorithm

Satisfy ve consistent kavramları nelerdir?

Satisfy sadece doğru durumlarda geçerli olmasının yeterli olmasıdır. Consistent ise hem doğru hem de yanlış durumlarda geçerli olmasıdır.

Inductive learning algoritmalarından ilki olan Find-S algoritması satisfy based algoritmadır.

Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

$h \leftarrow \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$

$h \leftarrow \langle Sunny, Warm, Normal, Strong, Warm, Same \rangle$ İlk datayı kendi datasına eşitliyor.

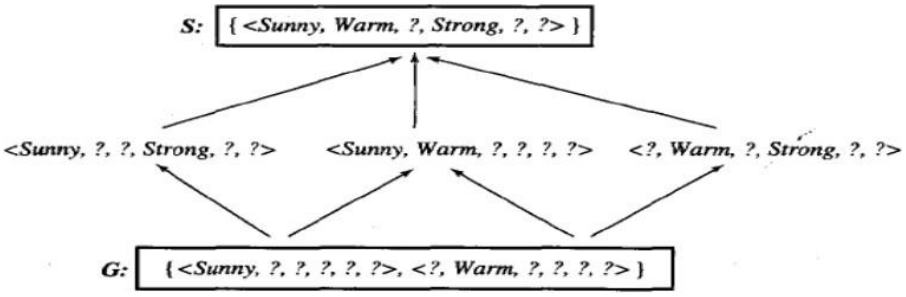
$h \leftarrow \langle Sunny, Warm, ?, Strong, Warm, Same \rangle$ Uyumluluk olmayan sahayı genellemiş.

$h \leftarrow \langle Sunny, Warm, ?, Strong, ?, ? \rangle$

- Veri içindeki hatalara karşı toleranslı değil.
- Kayıtların işleniş sırasına göre hızlıca tüm durumların ? olduğu duruma ulaşabilir. Hemen tıkanabilir.
- Alternatif sunmuyor yani version space bulmuyor.

2.2.2 List-Then-Eliminate Algorithm

Consistent based bir yaklaşımı benimser. -S algoritması özelden genele giderken bu algoritma genelden özele gider. Bu yaklaşım daha başarılı çalışır çünkü alternatif sunar ve version space bulur.



Bu algoritma da hatalara karşı dayanıklı değil fakat kayıtların işleniş sırasından etkilenmiyor.

Örnek: 6 saha olsun ve her saha 3 farklı değer alsın. Kaç farklı hipotez ifade edilebilir?

Version space: 2 üzeri 3 üzeri 6

Inductive learning kapsamında iki operatör bulunur.

?-değer

0

(3+1)^6 + 1

+1 soru işaretinden gelir

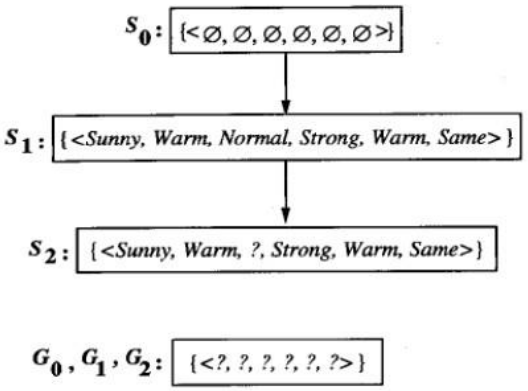
+1 yokluk kümesi

Yorum: Restrictive bias sayesinde çok büyük olan versiyon uzayından 4097 durum çıkarıldı.

$$\frac{(v+1)^i+1}{2^{v^i}}$$

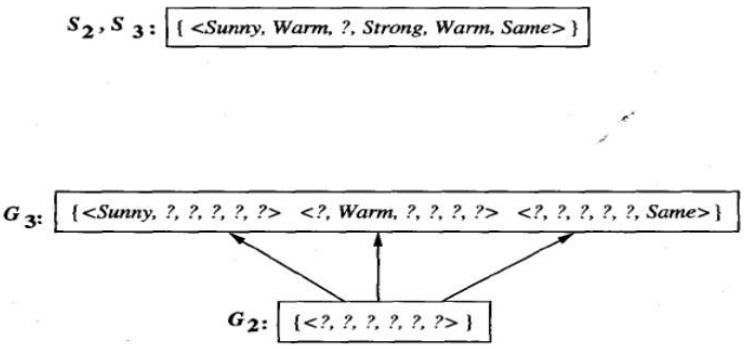
2.2.3 Candidate Elimination Algorithm

Find -S ile List-Then-Eliminate algoritmalarını beraber çalıştırır. Verinin geliş sırasından daha az etkilenir. Version space'in el olası daraltılmış halini bize sunuyor. (En ideal çözüm)



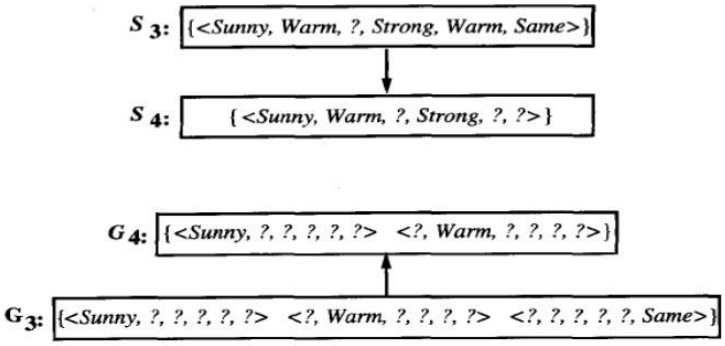
Training examples:

1. <Sunny, Warm, Normal, Strong, Warm, Same>, Enjoy Sport = Ye
2. <Sunny, Warm, High, Strong, Warm, Same>, Enjoy Sport = Yes



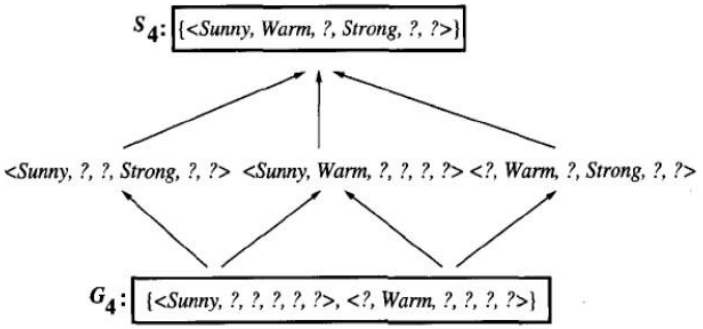
Training Example:

3. <Rainy, Cold, High, Strong, Warm, Change>, EnjoySport=No

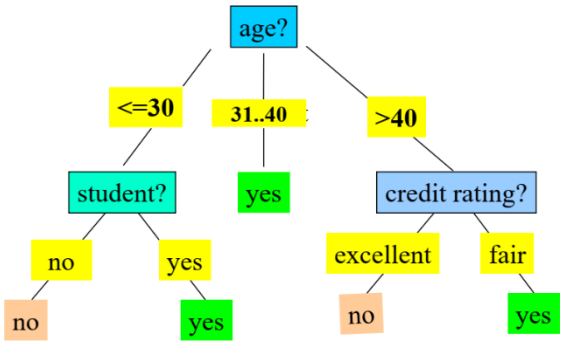


Training Example:

4. <Sunny, Warm, High, Strong, Cool, Change>, EnjoySport = Yes



3. Decision Trees



Frekans karar ağacının gücünü belirler. Örneğin yes durumundaki değerler 8 yes 2 no da olabilir 6 yes 4 no'da.

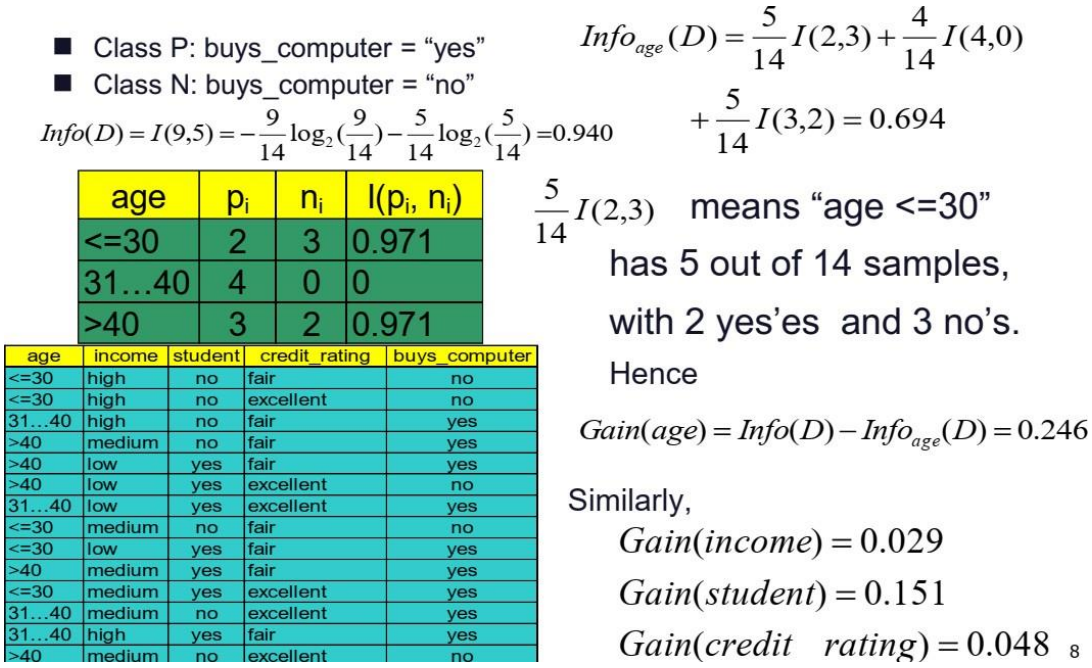
Bu nedenle data arttıkça frekans da artar ve hata oranı azalır.

Search bias'ın bir avantajıdır.

Durma koşulları 3 adettir.

1. Belirli bir node'daki tüm örnekler aynı sınıfa aittir.
2. Bölünecek başka attribute kalmamıştır.
3. Sample kalmamıştır.

3.1 Information Gain



age	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	0.971
31…40	4	0	0
>40	3	2	0.971

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

3.2 ID3 algoritmasının biasi: Küçük ağaçları her zaman büyük ağaçlara tercih etmektir.

ID3 algoritmasındaki amaç entropiyi en hızlı azaltacak yolu seçmektir. Küçük ağaç basit olduğundan karmaşıklığı önleyerek entropiyi daha kolay bir şekilde azaltır fakat dallanma sadece buna bağlı değildir ayrıca veri setinin de incelenmesi gerekir. Bazen büyük ve karmaşık ağaçların da çıkartılması gerekmektedir. Genelleme olarak bu yorum doğrudur fakat her zaman bu durum geçerlidir diyemeyiz.

Her zaman geçerli değildir. Search bias ile her adımda kısıtlama yapıyor. Information gain gibi heuristic yani sezgisel metotlar kullanıyor. Heuristic bize en optimal çözümü vermez fakat tüm ağaçları taramak çok zor olduğundan (neredeyse imkansız büyük verilerde) çok yararlı bir yöntemdir.

Farklı heuristic çeşitleri vardır. Örneğin bazı uygulamalarda üç adım daha ilerletilip karar verilir buna *windowing* denir. Window arttıkça daha doğru sonuç elde edilir fakat run time complexity'de artar. ID3 yaklaşımı tek heuristic değerlendirmesi yapar yani window kullanmaz.

3.3 Occam's Razor

Basit olanı tercih et. Karmaşık bir hipotez çok fazla şey ifade ettiğinden doğru çözümü de tesadüfen içine dahil etmiş olabilir. Yani karmaşık bir hipotez, tesadüfi sonuçlar doğurabilir.

3.4 Karar Ağaçlarında Overfitting

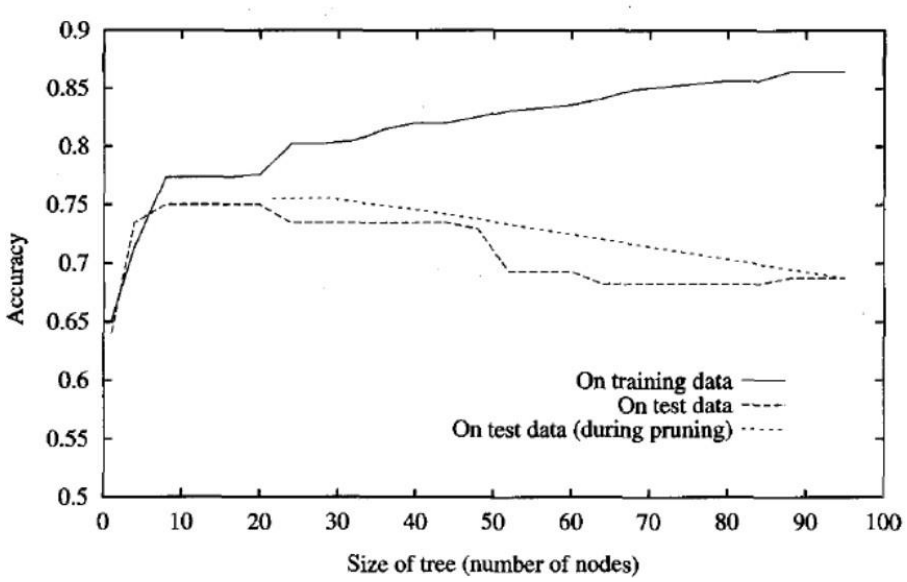
Genellikle noise içeren veride random hatalar bulunur ve algoritma bunları da açıklamak ister. Fakat temiz bir veride de gerçekleşebilir. Özellikle veri sayısı yetersizse ağacın alt dallarında frekans azalır ve bu nedenle bunun tesadüf mü olduğunu bilemeyiz.

Önlemek için 2 yöntem bulunur.

1. Ya ağacı en başta büyütmeyeceğiz. (pre prunning)
2. Ya da büyütüp budayacağız. (post prunning)

Pre prunning daha maliyetli olduğundan daha az tercih ediliyor. Çünkü test setini aynı anda çalıştırmak maliyet istiyor, uzun zaman alıyor çünkü her ağaç büyüdüğünde test setini çalıştırmamızı gerektiriyor.

Post prunning'de budama işlemi yapılırken frekanslardan yararlanılır.



3.5 C4.5 Gain Ratio

Information gain'in bias'ı aldatıcı olabilir. Çünkü sayıya bakmaz. Bazen bir ya da iki değerli durumlar tamamen pure olarak göründüğünden onları tercih etmek ister. Buna false purity denir. Yani çok değer olanları tercih ederler.

Bunu önlemek için C4.5 algoritması geliştirildi. **Penalty factor** ile çok değer alanları cezalandırır.

3.6 Gini Index

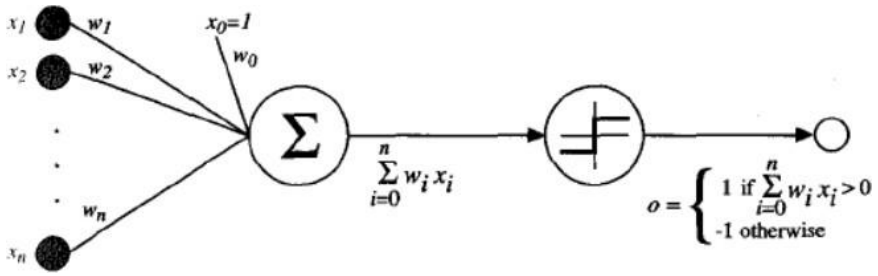
Direkt koşullu olasılık kullanılır. En önemli avantajı dengeli ağaçlar yaratma eğilimidir fakat sınıf sayısı fazla ise sıkıntılar yaşanır. Information gain'deki gibi çok değerli attribute'ları tercih ettiğinden gerçek hayatta gain ratio ile beraber kullanılmaktadır.

3.7 ID3 algoritmasının hipotez uzayı nedir?

Ağacın derinliği i, her node'da oluşabilecek dallanma v olmak üzere. Oluşabilecek en büyük ağacın node sayısı v^i olur. Alt küme sayısı 2 üzeri v^i olur. ID3 yalnızca bir çözüm sunduğundan hipotez uzayı 1'dir.

Bias ratio $1/(2\text{^}v\text{^}i)$ olur.

4. Neural Networks



Perceptron, en temel yapıdır. Özet, basitleştirme makinesi diyebiliriz. Perceptron'un mantığı basit olmasına rağmen bir XOR problemini çözememektedirler. 2 line gerekmektedir. Fakat birden fazla perceptron'un bağlanması bu problemi çözer. Öğrenme ağırlık değerleri güncellenerek gerçekleşir.

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta(t - o)x_i$$

t: olması gereken değer (target)

o: ürettiğimiz değer (output)

Amacımız aradaki farkı azaltmaktır. Önlerindeki çarpan learning rate'I ifade eder.

Delta training rule ya da diğer adıyla gradient descent tüm datadaki hata miktarını azaltmayı amaçlar.

$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$$

Her adımda en hızlı düşüşü garanti eder fakat totalde etmez. Top gibi düşünürsek sonlara doğru neredeyse hiç hareket etmeyebilir. Heuristic mantığına dayanır. Lineer düzlemlerde tek bir çözüm

yani global minimum bulunur. Local minimum bulunmadığından, locak minimum da takılmak gibi bir sorunu da yoktur.

$$\Delta \vec{w} = -\eta \nabla E(\vec{w})$$

Ağırlıklar E'nin türevi (gradient) alınarak güncellenir. Learning rate parametresi çok büyük olmamalıdır çünkü bu durumda minimumu atlayabilir. Genel yaklaşım adım sayısı arttıkça learning rate parametresini de küçültmektir.

Delta training rule her zaman perceptron öğrenmesinde hata sayısını minimize eder mi?

Deltra training rule heuristic mantığı ile çalışmaktadır. Yani bize en iyisi sonucu verememe riski de taşımaktadır. Bu nedenle her zaman hata sayısını minimize eder diyemeyiz. Eğer fonksiyonumuz lineer yapıda değilse lokal minimum noktalarında sıkışıp kalma özelliği de bulunmaktadır. Bu durumda çeşitli işlemlerle bu noktadan kurtulmak gerekir. Eğer bu işlemler gerçekleşmezse ve kurtulamazsak yine minimize etmediği sonucuna ulaşılırız.

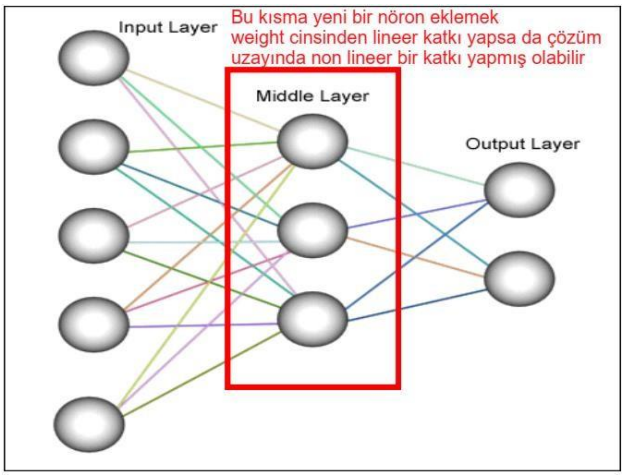
4.1 Stochastic vs. Gradient

Gradient Descent	Stochastic Gradient Descent
$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d)\, x_{id}$	$\Delta w_i = \eta (t - o)\, x_i$
$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$	$E_d(\vec{w}) = \frac{1}{2} (t_d - o_d)^2$

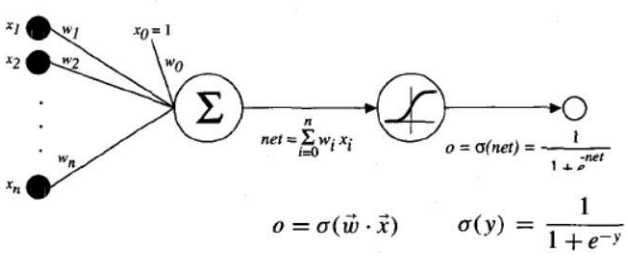
Stochastic birim hatayı ifade eder. Parametrelerini güncellemeden önce verinin tümünü dolaşmaz. Her bir örnekten sonra günceller. Daha hızlıdır fakat hata paya çok daha yüksektir. Büyük verilerde stochastic tercih edilir.

Stochastic gradient iki yönlü hareket edebilir. Yani süreci uzatıp salınımlar yapabilir. Lokal minimum problemlerinde o noktadan fonksiyonu kurtarabilir.

4.2 Multilayer Networks



Yeni bir middle layer eklemeden önce aynı layer'a nöron eklenmesi tercih edilmelidir çünkü her iki durum da non lineer bir katkı yapar fakat yeni bir layer eklemek daha masraflıdır. Fakat 2 adet middle layer ile tüm fonksiyonları ifade edebiliyoruz.



Perceptron'da kullanılmış sign fonksiyonu yerine sigmoid sürekli bir fonksiyon olduğundan türevi devam ettirebiliyoruz ve sayesinde non linearity elde ediyoruz. Bu da sign gibi değerleri sıkıştıran, küçülten kısaca özetleyen fonksiyondur. Bir önceki sayfada middle layer içerisinde 3 adet bu yapıdan bulunmaktadır.

4.3 Local Minima için Çözümler

1. Momentum term metriğinin ağırlık güncelleme fonksiyonunda kullanılması.

Bu fonksiyon, karar verme sürecinde sadece o an işlenmekte olunan dataya bakarak değil, önceki statelerde ki durumu da karar sürecine katmayı sağlamaktadır. Dataların göstereceği dağılımlar sonucunda o an işlenen dataların ani adaptasyon artışlarına sebep olmasını ve önceki

$$\Delta w_{ji}(n) = \eta\, \delta_j\, x_{ji} + \alpha \Delta w_{ji}(n-1)$$

statelerden ani sıçramalar yaparak uzaklaşmasını önlemektedir.

2. Gradient descent kullanırken stochastic gradient descent'a geçiş yapmak.

3. Birden fazla yapay sinir ağı eğitmek fakat hepsine başta random ağırlıklar atamak. Fakat bu çok masraflıdır ve gerçek hayatta çok tercih edilmez.

4.4 Penalty term

Weight'lerin çok büyük olması overfitting durumuna neden olacağından bu durumu önlemek istiyoruz ve cezalandırma yöntemi uyguluyoruz. Böylece daha dengeli ağırlıklar elde edilir.

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2 + \gamma \sum_{i,j} w_{ji}^2$$

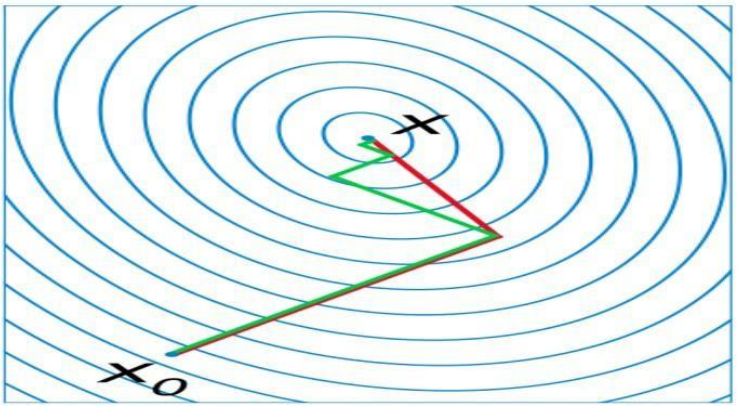
4.5 Cross entropy

$$-\sum_{d \in D} t_d \log o_d + (1 - t_d) \log(1 - o_d)$$

Bazen girdilerimiz sayılar sonucumuz ise olasılıksal olabilmektedir. Örneğin kredi başvurusunun verilmesi için gereken olasılık veya hastalığın olasılığı. Domain farklılığının önüne geçemk için cross entropy kullanılır. Farklı bir minimizasyon fonksiyonudur.

4.6 Conjugate gradient

Gradient descent yavaş olduğu için ikinci türevi kullanan conjugate gradient daha sık kullanılmaktadır. Hem büyük adımlarla hem de doğru yöne daha hızlı ilerlemeyi sağlar. Yeşil normal, kırmızı conjugent gradient'tir.



5. EKLER

5.1 Bayesian Learning

Amaç dataya en uyumlu hipotezi seçmek, aynı zamanda olasılığı da aktarmasını sağlamak. Version space oluşturuyor. Inductive learning'teki version space'den farklı hangi hipotezin daha iyi olduğunu olasılıklar yardımıyla söylemesidir. Hipotezleri sıralama imkanı sunmasıdır.

Merkezi limit teoremini kullanır. Minimum description length yaklaşımını benimser.

Occam's Razor ile Minimum Description Length teorilerini karşılaştırınız.

Occam's Razor perspektifinde bir çözümün olabildiğince basit olması gerektiği düşünülmektedir. Eğer bir çözümün daha basit bir yolu var ise o tercih edilmelidir. Çözümü basitleştirmek, karar verme mekanizmasının daha basit yollarla karar vermesine ve karmaşık ilişkilerden kurtardığı içinde overfitting riskinden uzaklaştırmaktadır.

Minimum Description Length Information Teorisi ise Occam's Razor perspektifinde eksik olan, *elimizdeki veriye uyumlu olma* çabasını hesaplamalara katılmış halidir.

Veriler üzerinden öğretmek istediğimiz modellerde, veri ile uyumluluğun kontrol edilmemesi durumunda model başarısızlıkla sonuçlanacaktır. Minimum Description Length Information Teorisi, Occam's Razor perspektifine göre zıt bir perspektif değildir. Eksik olduğu noktalara yeni bir bakış açısı getirmiş ve Occam's Razor perspektifinde olduğu gibi hala olan en basit çözümü aramaktadır

5.2 Gibbs Algoritması

Bayes optimal classifier hipotezlere ağırlıklar verir ve ağırlıklara göre sıralama yapar. Fakat birbirine yakın hipotezlerde hata yapmamak için her birini dener. Gibbs ise hepsine bakmaz, en uygun algoritmanın görüşünü alır.

5.3 Öğrenme Çeşitleri

1. Aktif

Anlık gelişen ve gördükçe öğrenen yapılardır. Model sürekli yaşamaktadır. Öğrenme ve sınıflandırma maliyeti dengededir.

2. Pasif

Arka planda öğrenir ve modellenir. Model devamlı yaşamaz yeni yeni modeller oluşturulur. Öğrenme maliyeti yüksek, sınıflandırma düşük.

3. Lazy

Yalnızca yeni gelen instancelara ve ona yakın olanlara bakar. Mesela 2 tane + durum var fakat yeni gelen örneğe en yakın – ise sonucun – olma olasılığı daha fazladır diyip daha yüksek bir ağırlık veriyor. Öğrenme maliyeti düşük fakat sınıflandırma maliyeti yüksektir.

Lazy learning'de karşılaşılan Curse of Dimensionality nedir?

Seçilen attribute'ların yani dimensionların sayısı (çok fazla ise) ve niteliği doğru seçilmez ise karşılaşılan bir problemdir. Learningde hesaplanmak istenen sonuçlarla alakasız veya risk oluşturabilecek attributeların kullanılması, iki noktanın birbirine olması gerektiğinden daha yakın veya daha uzak hesaplanmasına sebep olabilir. Bu durum algoritmanın düzgün çalışmasını önler.

Bu duruma sebep olan bir başka şey ise "Cinsiyet" gibi çok az değer kümesi barındıran attributelardır. Bu tarz attributelar iki node arasında yeterli değerde farklılığın oluşmasını engelleyebilmektedir.