Yapay Zeka Dersi için Çalışma Notları

1. Yapay Zeka (Genel Bir Bakış)

- **Yapay Zeka Tanımı**: Bilgisayar sistemlerinin, insan zekâsı gerektiren görevleri yerine getirebilme yeteneği.
- **AI'nin Alt Alanları**:
  - **Makine Öğrenmesi**: Veriden öğrenme ve tahmin yapma.
    - **Gözetimli Öğrenme**: Etiketli veri kullanarak model oluşturma.
    - **Gözetsiz Öğrenme**: Etiketlenmemiş veriden yapıları keşfetme.
    - **Pekiştirmeli Öğrenme**: Çevre ile etkileşime girerek öğrenme.
  - **Doğal Dil İşleme**: İnsan dilini anlama ve işleme.
    - **Dil Modellemesi**: Metin verilerini anlamak ve üretmek.
    - **Dil Çevirisi**: Diller arasında çeviri yapma.
    - **Duygu Analizi**: Metinlerdeki duygusal içeriği belirleme.
  - **Bilgisayarla Görme**: Görsel verileri işleme ve yorumlama.
    - **Görüntü Sınıflandırma**: Görselleri kategorilere ayırma.
    - **Nesne Tespiti**: Görsellerdeki nesneleri tanımlama.
    - **Görüntü Segmentasyonu**: Görüntüleri bölgelere ayırma.
  - **Otonom Sistemler**: Kendi kendine hareket edebilen sistemler.
    - **Robotik**: Fiziksel robotların kontrolü ve hareket planlaması.
    - **Otonom Araçlar**: Kendi kendine gidebilen araçlar.

2. Mantığa Giriş

- **Mantık**: Argümanların geçerliliğini belirleme bilimi.
- **Argümanlar**: Önermelerden oluşan ve bir sonuca ulaşmayı amaçlayan cümleler dizisi.
  - **Önermeler**: Doğru veya yanlış olabilen ifadeler.
  - **Sonuç**: Önermelerden türetilen ifade.
- **Geçerlilik**: Eğer önermeler doğruysa, sonuç da doğru olmalıdır.
- **Seslilik**: Geçerli bir argümanın tüm önermeleri doğrudur.

3. Önerme Mantığı ve Mantıksal Operatörler

- **Önerme**: Doğru veya yanlış olabilen ifade.
- **Mantıksal Operatörler**:
  - **Ve (∧)**: Her iki önerme de doğruysa, sonuç doğru.
  - **Veya (∨)**: En az bir önerme doğruysa, sonuç doğru.
  - **Değil (¬)**: Önerme doğruysa, sonuç yanlış; önerme yanlışsa, sonuç doğru.
  - **İse (→)**: İlk önerme doğru ve ikinci önerme yanlışsa, sonuç yanlış; diğer durumlarda doğru.
  - **Ancak ve Ancak (↔)**: Her iki önerme aynı doğruluk değerine sahipse, sonuç doğru.
- **Örnekler**:
  - **Ve (∧)**: "Ali ve Ayşe çalışkandır" ifadesi, Ali ve Ayşe'nin ikisi de çalışkan olduğunda doğrudur.
  - **Veya (∨)**: "Ali veya Ayşe çalışkandır" ifadesi, Ali veya Ayşe veya ikisi birden çalışkan olduğunda doğrudur.
  - **Değil (¬)**: "Ali çalışkan değil" ifadesi, Ali çalışkan olmadığında doğrudur.
  - **İse (→)**: "Eğer Ali çalışkansa, Ayşe de çalışkandır" ifadesi, Ali çalışkan olup Ayşe çalışkan olmadığında yanlıştır.
  - **Ancak ve Ancak (↔)**: "Ali ancak ve ancak Ayşe çalışkansa çalışkandır" ifadesi, Ali ve Ayşe aynı doğruluk değerine sahip olduğunda doğrudur.

4. Önerme Mantığının Sembolik Gösterimi ve Sentaks

- **Sembolik Gösterim**: Doğal dildeki ifadelerin sembollerle gösterilmesi.
  - **Örnek**: "Eğer yağmur yağıyorsa, zemin ıslaktır" ifadesi, p→qp \rightarrow qp→q olarak gösterilir.
- **Temel İfadeler**:
  - **Mantıksal Sabitler**: ¬, ∧, ∨, →, ↔
  - **Önerme Değişkenleri**: p, q, r, vb.
- **Oluşum Kuralları**:
  - Herhangi bir önerme değişkeni bir formüldür.
  - φ bir formülse, ¬φ de bir formüldür.
  - φ ve ψ formülleriyse, (φ ∧ ψ), (φ ∨ ψ), (φ → ψ), (φ ↔ ψ) de formüldür.
- **Doğruluk Tablosu**: Her bir mantıksal operatörün doğruluk değerlerini gösteren tablo.
  - **Örnek Doğruluk Tablosu**:
    - ppp: 0, 0, 1, 1
    - qqq: 0, 1, 0, 1
    - p∧qp \land qp∧q: 0, 0, 0, 1

5. Semantik ve Anlam

- **Semantik**: Dilin anlamını belirleme.
- **Doğruluk Değerleri**: Önerme mantığında anlam, bir ifadenin doğruluk değeriyle belirlenir.
    - **Örnek**: "Ali evdedir" ifadesi doğruysa, semantik değeri 1 (doğru) olur.
- **Doğruluk Tablosu**: Bileşik ifadelerin doğruluk değerlerini belirlemek için kullanılır.
    - **Örnek**:
        - p→qp \rightarrow qp→q ifadesi için doğruluk tablosu:
            - ppp: 0, 0, 1, 1
            - qqq: 0, 1, 0, 1
            - p→qp \rightarrow qp→q: 1, 1, 0, 1

6. Önerme Mantığında Geçerlilik

- **Geçerlilik**: Bir argümanın geçerli olması için, tüm modellerde (doğruluk tablosundaki tüm satırlarda) önermeler doğru olduğunda sonuç da doğru olmalıdır.
    - **Örnek**: "Eğer Ali çalışkansa ve Ayşe de çalışkansa, Ali ve Ayşe çalışkandır" ifadesi geçerlidir.
- **Karşı Model**: Önermelerin hepsinin doğru, ancak sonucun yanlış olduğu bir model.
    - **Örnek**: p∧q→rp \land q \rightarrow rp∧q→r ifadesi için karşı model, p∧qp \land qp∧q doğru olduğunda rrr yanlışsa var olur.

7. Önerme Mantığında Doğruluk Tabloları

- **Doğruluk Tablosu Oluşturma**: Her bir önerme değişkenine çeşitli doğruluk değerleri atayarak tüm olası modelleri oluşturma.
    - **Örnek**: p→(q∧r)p \rightarrow (q \land r)p→(q∧r) ifadesi için doğruluk tablosu:
        - ppp: 0, 0, 0, 0, 1, 1, 1, 1
        - qqq: 0, 0, 1, 1, 0, 0, 1, 1
        - rrr: 0, 1, 0, 1, 0, 1, 0, 1
        - q∧rq \land rq∧r: 0, 0, 0, 1, 0, 0, 0, 1
        - p→(q∧r)p \rightarrow (q \land r)p→(q∧r): 1, 1, 1, 1, 0, 1, 0, 1

8. Önerme Mantığında Türetme ve Doğruluk

- **Sentaktik Geçerlilik**: Mantıksal kurallar kullanılarak bir argümanın geçerliliğinin belirlenmesi.
    - **Örnek**: Modus Ponens (Doğrulama Yolu):
        - p→qp \rightarrow qp→q
        - ppp
        - qqq
- **Doğal Türetim**: Argümanların geçerliliğini belirlemek için kullanılan formal yöntem.
    - **Örnek**: Doğruluk tabloları kullanarak geçerlilik kontrolü.

9. Birinci Derece Önerme Mantığı (First-Order Predicate Logic)

- **Gereklilik**: Daha karmaşık ifadeleri ve argümanları ifade etmek için gereklidir.
- **Yeni Diller**: L1'in eksikliklerini gidermek için yeni formel diller geliştirilmiştir.
- **Kuantörler**: Evrensel (∀) ve varlıksal (∃) kuantörler kullanılır.
    - **Örnekler**:
        - **Evrensel Kuantör (∀)**: "Her öğrenci çalışkandır" ifadesi, ∀x (öğrenci(x) → çalışkan(x)) olarak gösterilir.
        - **Varlıksal Kuantör (∃)**: "Bazı öğrenciler çalışkandır" ifadesi, ∃x (öğrenci(x) ∧ çalışkan(x)) olarak gösterilir.
- **L2 Dili**: Birinci dereceden mantık için kullanılan yeni bir dil.
- **Örnekler**:
    - "Her insan ölümlüdür" ifadesi, ∀x (insan(x) → ölümlü(x)) olarak gösterilir.
    - "Bazı insanlar doktorlardır" ifadesi, ∃x (insan(x) ∧ doktor(x)) olarak gösterilir.

10. Önerme-Argüman Analizi

- **Dil ve Mantık**: Dilin sınırları, dünyamızın sınırlarını belirler.
- **L1 ve L2**: L1 dilinin eksiklikleri ve L2'nin gerekliliği.
- **Kuantör Kullanımı**: Evrensel ve varlıksal kuantörlerin kullanımı.
- **Örnek**: "Ali'nin annesi veya babası vardır" ifadesi, ∃x (anne(x, Ali) ∨ baba(x, Ali)) olarak gösterilir.

11. İlk-Order Mantıkta Programlama

- **Mantıksal Programlama**: Durumların ve olguların mantıksal olarak ifade edilmesi.

- **Kural ve Gerçekler**: Gerçekler mevcut durumu tanımlar, kurallar ise türetme işlemlerini tanımlar.
  - **Örnek**:
    - **Gerçek**: insan(Ali).
    - **Kural**: ölümlü(X) :- insan(X).
- **Recursive Programlama**: Kendi kendini çağıran kurallar kullanılarak daha genel ve kapsamlı programlar yazma.
  - **Örnek**: atası(X, Y) :- ebeveyni(X, Y). atası(X, Z) :- ebeveyni(X, Y), atası(Y, Z).

12. Recursive Programlama

- **Özyineleme (Recursion)**: Bir prosedürün kendisini çağırarak tanımlanması.
- **Temel ve Özyinelemeli Kural**: Temel kural, durumu belirler; özyinelemeli kural ise daha karmaşık durumları tanımlar.
  - **Örnek**:
    - **Temel Kural**: faktöriyel(0, 1).
    - **Özyinelemeli Kural**: faktöriyel(N, F) :- N > 0, N1 is N - 1, faktöriyel(N1, F1), F is N * F1.
- **Özyinelemeli Yapılar**: Sınırsız sayıda işlemi ifade etmek için kullanılır.
  - **Örnek**: Doğal sayıların tanımı ve aritmetik işlemler.

13. Prolog'da Listeler

- **Liste Yapısı**: Baş (head) ve kuyruk (tail) bileşenlerinden oluşur.
  - **Örnek**: [a, b, c, d] listesi, a başı ve [b, c, d] kuyruğuna sahiptir.
- **Listelerle İlişkiler**: Prolog'da çeşitli ilişkiler listeler kullanılarak tanımlanabilir (örneğin, eleman, alt liste, tersine çevirme).
  - **Örnek**:
    - **Eleman**: member(X, [a, b, c]).
    - **Alt Liste**: sublist([b, c], [a, b, c, d]).
    - **Tersine Çevirme**: reverse([a, b, c], [c, b, a]).

14. Aritmetikte Özyineleme

- **Doğal Sayılar**: 0 ve ardışık sayılar kullanılarak tanımlanır.
  - **Örnek**: doğal(0). doğal(s(N)) :- doğal(N).
- **Aritmetik İşlemler**: Toplama, çarpma ve üslü ifade gibi işlemler özyinelemeli olarak tanımlanabilir.
  - **Örnek**:
    - **Toplama**: toplama(0, Y, Y). toplama(s(X), Y, s(Z)) :- toplama(X, Y, Z).
    - **Çarpma**: çarpma(0, _, 0). çarpma(s(X), Y, Z) :- çarpma(X, Y, W), toplama(W, Y, Z).
    - **Üslü İfade**: üs(_, 0, s(0)). üs(X, s(Y), Z) :- üs(X, Y, W), çarpma(X, W, Z).
- **Ackermann Fonksiyonu**: Hesaplanabilir ama ilginç bir örnek olan Ackermann fonksiyonu.
  - **Örnek**: ackermann(0, N, s(N)). ackermann(s(M), 0, A) :- ackermann(M, s(0), A). ackermann(s(M), s(N), A) :- ackermann(M, R, A), ackermann(s(M), N, R).

15. Birleştirme (Unification)

- **Birleştirme Operasyonu**: İki terimi eşitleme işlemi.
  - **Örnek**: X = Y, X = a, Y = a.
- **Yedekleme (Substitution)**: Değişkenlerin değerlerle değiştirilmesi.
  - **Örnek**: {X = a, Y = b}, {Y = c}.
- **Yönsüzlük**: Birleştirme işlemi yönsüzdür, yani terimlerin yerleri değiştirilebilir.
  - **Örnek**: X = a ve a = X aynı sonuçları verir.

16. Mantıksal Yorumlayıcı (Interpreter)

- **Yorumlayıcı**: Sorguları ve programları yorumlayarak mantıksal sonuçlar çıkarır.
  - **Örnek**: Prolog yorumlayıcısı, bir sorguyu alır ve ilgili kuralları kullanarak sonucu hesaplar.
- **Kontrol Mekanizması**: Geri izleme stratejisi ile çalışır.
  - **Örnek**: Derinlik öncelikli arama stratejisi kullanarak çözüm bulma.
- **Örnek Program**: Prolog'da aile ilişkilerini modelleyen bir program ve bunun çözüm ağacı.
  - **Örnek**:
    - **Program**: ebeveyn(ali, ayşe). ebeveyn(ayşe, mehmet).
    - **Sorgu**: atası(ali, mehmet).

17. Meta Programlama

Meta programlama, programların kendilerini veya diğer programları manipüle edebilme yeteneğine sahip olduğu bir programlama tekniğidir. Meta programlama ile yazılan programlar, kodun çalışma zamanında dinamik olarak değiştirilebilmesini sağlar.

Örnek 1: Basit Bir Meta Programlama Kullanımı

Bir meta program, diğer programları veya kod parçalarını işleyebilir ve bunlar üzerinde değişiklikler yapabilir. Örneğin, bir Prolog programında dinamik olarak yeni kurallar ekleyebiliriz:

```
prolog
Kodu kopyala
% Normal Prolog kuralları
ebeveyn(ali, ayşe).
ebeveyn(ayşe, mehmet).

% Dinamik olarak yeni kural eklemek
:- dynamic(ebeveyn/2).

% Yeni ebeveyn ilişkisi eklemek
assertz(ebeveyn(mehmet, ahmet)).
```

Bu örnekte, `assertz/1` predikatı kullanılarak çalışma zamanında yeni bir ebeveyn ilişkisi eklenmiştir. Bu, programın dinamik olarak genişletilebilmesini sağlar.

Örnek 2: Meta Programlama ile Sorgu Çözümleme

Meta programlama, bir programın kendi kodunu analiz edebilmesini ve bu kodu işleyebilmesini de sağlar. Örneğin, bir Prolog programında belirli bir sorgunun nasıl çözüldüğünü analiz edebiliriz:

```
prolog
Kodu kopyala
% Örnek kurallar
büyükbaba(X, Z) :- ebeveyn(X, Y), ebeveyn(Y, Z).

% Meta programlama ile çözümleme
çöz(Sorgu) :-
    call(Sorgu),
    format('Sorgu başarılı: ~w~n', [Sorgu]).
çöz(Sorgu) :-
    format('Sorgu başarısız: ~w~n', [Sorgu]).

% Örnek sorgu
?- çöz(büyükbaba(ali, mehmet)).
```

Bu örnekte, `call/1` predikatı kullanılarak verilen sorgu çalıştırılır ve sonucu analiz edilir. Eğer sorgu başarılı olursa, başarılı olduğu mesajı yazdırılır; başarısız olursa, başarısız olduğu mesajı yazdırılır.

Örnek 3: Dinamik Kuralların Silinmesi

Meta programlama ile dinamik olarak eklenen kurallar da silinebilir. Bu, programın esnekliğini ve uyarlanabilirliğini artırır:

```
prolog
Kodu kopyala
% Dinamik kural eklemek
assertz(ebeveyn(ali, veli)).

% Dinamik kuralı silmek
retract(ebeveyn(ali, veli)).

% Kuralları listelemek
listing(ebeveyn/2).
```

Bu örnekte, `assertz/1` ile eklenen dinamik bir kural, `retract/1` ile silinir ve `listing/1` ile mevcut tüm ebeveyn ilişkileri listelenir.

Meta Programlama Avantajları

1. **Esneklik**: Programların çalışma zamanında dinamik olarak değiştirilmesini sağlar.
2. **Kod Yeniden Kullanımı**: Genel ve soyut işlemler tanımlanarak kodun daha kolay yeniden kullanılmasını sağlar.
3. **Dinamik Adaptasyon**: Programların çalışma ortamlarına dinamik olarak uyum sağlamasına olanak tanır.

Meta programlama, özellikle yapay zeka ve mantıksal programlama alanlarında güçlü bir araçtır. Programların daha esnek ve uyarlanabilir olmasını sağlar ve karmaşık sorunların daha kolay çözülmesine yardımcı olur.