Yapay zekaya giriş

Hisar CS.
hisar computer science

İçindekiler

Emre Can

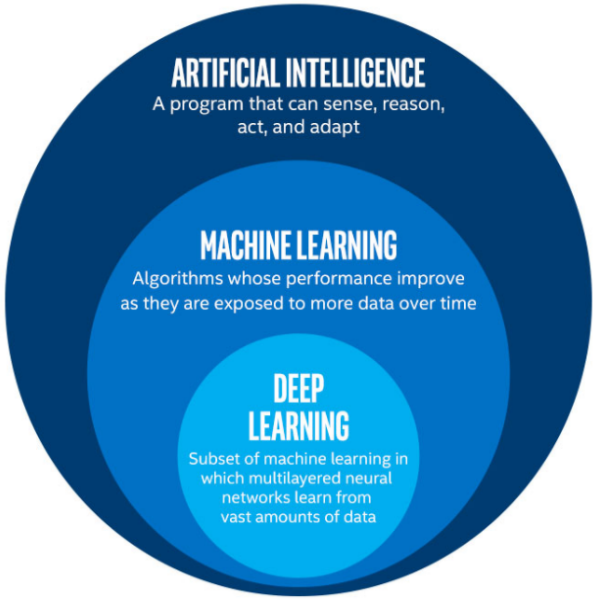
Giriş

Heryerde gördüğümüz ve yüksek vaatlerle hayatımıza giren yapay zeka şu anda bilgisayar dünyasının en ileri gelen araştırma konusudur. Buna rağmen alanın en ileri gelen isimlerinin bile cevaplayamadığı sorular var. Peki bu yeni teknoloji nelere tekabül? Yapay zeka bizim gibi düşünüp hissedebilir mi?

Yapay Zeka

Yapay zeka nedir?

Yapay zeka her türlü bilinçli davranışla ilgilidir. Bir nevi insanlar gibi yeni, zeki varlıklar yaratmak amacıyla geliştirilen bir teknolojidir. Zeki burada hızlı işlem yapabilen veya fazla bilgi tutabilen bir makinenin yerine, öğrenebilen adapte olabilen ve gelişen bir zeka anlamına gelir.
Burda önemli bir ayrım yapmamız gerekir. "Makine öğrenmesi" ve "yapay zeka" farklı alanlardır. Yapay zeka aslında makina öğrenmesinin üst başlığıdır. Bildiğimiz algoritmalar veya kodların aksine kendi kendine öğrenebilen algoritmalara makine öğrenmesi denir. Yapay zeka ise bu algoritmaların bir alanda toplanarak, birlikte çalışması sonucunda düşünebilen ve öğrenebilen bir varlık ortaya çıkarılmasıdır.



Yapay zekanın ana ilham kaynağı doğadan gelir. Bilim adamları şu anda doğada bulunan zeki varlıkların nasıl ve hangi koşullar altında nasıl davrandıklarını bilgisayarlar tarafından kopyalanmasını sağlamaya çalışmakatalar. Doğada bulunan inanılmaz kompleks beyin yapılarını kopyalayarak bir bakıma dijital bir beyin ortaya çıkarmak için çalışmaktalar.

Makine Öğrenmesi

En yaygın olarak kullanılan "makine öğrenimi", "yapay zeka" ve "derin öğrenme" arasındaki farklara göz atalım. Az önce de bahsettiğimiz üzere makine öğrenimi yapay zekanın üst başlığıdır. Makine öğrenim adı verilen kavram: bilgisayarların her değeri ve özelliği insan tarafından kodlanmadan verilen veriler ve gerekli ise insan giriş ile, ileride tahmin etme özelliğine sahip bir "hipotez" yaratabilmesidir. Adının verildiği gibi makinaların verilen girdi ile öğrenmesidir yani.

Yapay Zeka

"Yapay zeka" dediğimiz alan ise bu makine öğreniminin özel bir alt bölümde kullanılarak zeki ve daha özelleştirilmiş bir program yaratmak amacıdır. Bu programlar tıpkı insanlar gibi mantık yürütme yetisine sahip şekilde geliştirilmekte, ama kim bilir gelecekte neler olabilir.

Deep Learning

"Deep learning" ise aynı şekilde makine öğrenmesinin bir alt başlığıdır. Bu alanda bir nevi doğada bulunan beyinlerin kullandığı "nöron" sisteminin kopyalanması üzerine kuruludur. Bu tür algoritmalar tamamen insandan bağımsız olarak yalnızca verilen veri ile kendisi bir çıkarım yapması için veya insan denetimi ile öğrenip ileride verilen verilerde de bir tür çıkarımlar veya tahminler yapmasını sağlamaktır.

Yapay Zeka kullanım alanları

Yapay Zeka kullanım alanları

Her gün kullandığımız onca uygulamada şu anda yapay zeka uygulamaları kullanılıyor. Facebookta ön sayfanıza gelen video ve haberler. Netflix'te size önerilen film ve diziler. Youtube da önceki seçeneklerinize göre size özel bir ana sayfa ve yan bar oluşturma. Yapay zekanın kullanım alanları şu anda inanılmaz geniş. Farkında olmasanız bile sürekli olarak veri yüklediğimiz bu algoritmalar bazıları tarafından korkulsa da, aslında bize yardımcı olabilir. Şu anda kadar kullanılan yapay zeka uygulamaları filmlerde görülenlerin aksine dünyayı ele geçirmeye çalışan ileri makinalar durumunda değiller. Şu an yalnızca sizin nereden alışveriş yaptığınızı ve zevklerinizi anlayarak size özel bir servis vermekte yardımcı olurlar.

Yapay zeka geçmişi

İkinci dünya savaşının sonlarında ortaya çıkan Nazi savaş makinasının gizlice hareket etmesini sağlayan enigma makinesi zamanın tüm bilim insanları ve tasarımcılarının kafasını karıştırmıştı. Alan Turing'in yaptığı "Bombe" makinası ile kırılan Enigma kodları tarihin yönünü değiştirmiştir. Makine öğrenmesinin temelleri bu iki makina ile atıldığı söylenebilir. Alan Turing'e göre bir insan ve makine karşılaştırılırsa, makinelerin "taklit oyununu" kazanacağını öne sürmüştür, ve bu nedenle de akıllı birer varlık olabileceklerini söyler.

1956 da, John McCarthy tarafından gerçekleştirilen "Dartmouth konferansı" sırasında dönemin en ileri gelen bilgisayar mühendislerinin katıldığı konferansta ortaya "yapay zeka" kavramı ortaya çıkar. Allen Newell ve Herbert Simon, yapay zekanın ilerletilmesi için öncüler olarak araştırmalara destek ve yeni araştırmacılar sağlamışlardır. Bunun yanında yapay zekanın gelişmesi için tanıtılmasına yardım etmişlerdir.

1951 yılında ortaya çıkan "Ferranti Mark 1" algoritma kullanarak dama oynamayı öğrenmiş ve ustalaşmıştır. Newell ve Simon sonrasında matematiksel problemler için bir genel sorun çözücü algoritma tasarlamayı da başardılar. Yine aynı yıllarda John McCarthy "LISP" adı verilen bir kodlama dili ortaya çıkardı. Bu dil ileride makine öğrenmesi için önemli hale gelecektir.

1960 yıllarında araştırmacılar matematiksel sorun ve geometric teorileri çözecek algoritmalar üzerinde yoğunlaştılar. 1960'ın sonlarına doğru "makine görüşü" üzerinde çalılmanın yanında öğrenebilen robotlar ortaya çıkarmaya çalıştılar. "WABOT-1" ilk "akıllı" humanoid robot olarak 1972 Japonyada ortaya çıktı.

Maalesef bütün araştırma ve gelişmelere rağmen araştırmacılar makinelere akıl vermeyi başaramadı. Bunun en büyük sebebi dönemin yetersiz bilgisayarları ve işleme güçleriydi. Makine görüşü gibi uygulamaların doğru ve efektif bir şekilde çalışabilmesi için inanılmaz boyutta veriyi gözden geçirmesi gerekiyordu. Yavaşlayan ilerlemelerden dolayı hem genel halk hem de devlet yapay zekadan umudunu yitirmeye başlamıştı. Bu sebepten dolayı 1970-1990 yılları yapay zeka için "AI Wınter" denilen bir duraklama çağından geçmiştir. Azalan ilgi ve kaynaklardan dolayı bu dönemlerde araştırmacıların kayda değer bir bulguları yoktur. Ama bunların hepsi bilgisayarların iyice gelişmesi ile değişecekti.

1990'ların sonuna doğru yapay zekaya duyulan ilgi tekrardan artmaya başladı. Japon devleti yapay zekayı ilerletebilmek amacıyla beşinci jenerasyon yeni bilgisayarlar üstünde çalıştıklarını açıkladıktan sonra yapay zeka fanatikleri bilgisayarların yakında dilleri çevirebileceğini, bizimle konuşabileceklerini veya resimlere anlam verebileceklerine inanmaya başladılar. 1997 yılında IBM'in "Deep Blue" bilgisayarı, zamanın satranç dünya şampiyonu olan Garry Kasparov'u yenmeyi başardı.

Son zamanlarda zaten yapay zekanın ne kadar yol kat ettiğini her yerde görebiliyoruz. Son 15 yılda Google, Amazon, microsoft gibi birçok firma hem kullanıcıların davranışlarını anlamak için, hem de yapay zeka alanına katkıda bulunmak için birçok atılım ve gelişme yaptı. Son zamanlarda bu teknolojinin halka da açılması yeni nesillerin yapay zekayı sınıf içinde kullanmasına ve aynı zamanda araştırmalara katkıda bulunmasını kolaylaştırdı. Geleceğin ne getireceğini ancak hayal edebiliriz.

getireceğini ancak hayal edebiliriz.

Machine Learning Concepts

Giriş/Açıklama:

Makine öğrenimi olarak adlandırdığımız yöntem yapay zekanın alt kategorilerinden biridir. Daha önce üstünden geçtiğimiz makina öğrenimini burada daha detaylı bir şekilde inceleyeceğiz. Makine öğrenimin en temelinde matematik ve istatistik bulunur. Statistik kullanarak verilen veriden belirli trendleri çıkararak gelecek verilerin içinden tahmin yapmalarını sağlarız. Yani bilgisayarların en büyük sorunu olan her bir öğenin elle kodlanması sorununu en azından bu tür aplikasyonda ortadan kaldırabiliriz. Basitleştirmemiz gerekirse, makina öğrenimi bir bebeğe resimlerden şekilleri, veya duyduğu seslerin neye ait olduğunu öğretmeye benzer. Yüzlerce veya binlerce kez farklı aslan resimleri ve sesleri dinleterek bu çocuğa neyin bir aslan ve neyin aslan olmadığını öğrenmesi gibi.

Makineler nasıl öğrenir?

Az önce bahsettiğimiz üzere makine öğrenmesi aslında bir istatistik oyunudur. Algoritmaları kullanarak yüzlerce veri parçasının bu algoritmalardan geçirilerek, belirli trendler, ve benzerliklerin bulunması üzerine kuruludur. Aslında bir algoritmanın içinde bulunan ve değişen değerleri o algoritmayı yapan insanlar bile çoğu zaman bilmiyor. Aslında bu iyi birşey, özellikle karmaşık ve çok farklı sınıflama yapabilen algoritmaların her bir küçük değerini elle değiştirip bulmaya çalışmak neredeyse imkansız, ve başarılsa bile tamamen şansa bağlı.
Makine öğrenme algoritmalarının birçok belirli terminolojisi bulunmakta. Bunların her biri bu algoritmaların istendiği şekilde çalışması için vazgeçilmez özellikleri barındırıyor.

Training and Testing

Bu bölüm adından da kolayca anlaşılabilir. Algoritmaların gelecek veriden çıkarım yapıbilmesi için öncelikle bu veriden ne çıkarabileceğini öğrenmesi gerekir. Yani sürekli olarak penguen resimleri ile eğitilmiş algoritmalardan bir insanın kim olduğunu bulmayı, veya bir aslanı tanımlamasını bekleyemezsiniz. “Training” adı verilen işlem bu algoritmalara fazla sayıda resim, sayı, veya genel olarak herhangi bir tür veri verilmesi ve bu verilerden belirli özelliklerin çıkarılması üzerine kuruludur. Burada algoritmamızı eğitiriz. Bu eğitimin ardından, algoritma kendisi içinde belirli ağırlıklar ve trendler bularak belirli çıkarımlar yapar. Hatta çoğu zaman algoritmaları yazan insanların bile kendilerinin takip edemediği bir adımdır training. Bu hem iyi hem kötü birşey. Iyi yanı insanların resimlerden belirli sonuçları çıkarabilmeleri için her bir değişkeni çok ama çok küçük adımlarla değiştirmelerini, veya resimlerdeki beklenen öğelerin elle kodlanması gereğini ortadan kaldırır. Kötü yanı ise elimize içinde ne olduğunu bilmediğimiz bir kapalı kutu ortaya çıkar. Ne kadar doğru tahmin edeceğini veya nerede neyin yanlış gitmiş olabileceğini bilemeyiz.
İşte bu sebepten dolayı eğittiğimiz algoritmaları deneriz. Bu işleme “Testing” adı verilir. Testing, algoritmanın verilen yeni verilerin üstünde ne kadar doğru tahmin yapabileceğini görmemizi sağlar. Bu işlem aslında gayet basittir: algoritmaya daha önce görmediği bir veri üstünde tahmin yapmasını isteriz, ardından bu verinin doğru cevabına ne kadar yakın olduğunu ölçeriz. Bir nevi algoritmamaları bir sınava sokmak gibi düşünülebilir. Bu adım için elimizde sayılı veri varsa, bu verinin bir kısmını training, bir kısmını ise test için ayırmamız gerekli. Algoritmamızda birşey yanlış giderse test işlemini uygulayarak, tekrar train etmeyi deneyebilir, veya uygulamalarımızda kullanmaya başlayabiliriz.

Supervised Learning

Aslında şu ana kadar sürekli olarak supervised learning hakkında bahsettik. Supervised learning

yöntemi algoritmalara bir giriş verisi, ve belirli bir çıkış yolu verilmesi yoluyla çalışır. Yani algoritmaların çalışma aslı giriş verisinin belirli sonuca ulaştırmaya çalışarak çalışır. Bir test kitabının cevap anahtarını kullanarak çözdükten sonra, daha sonra gördüğünüz soruları anahtar olmadan çözmekle aynı şeydir. Bu tür algoritmaların en geniş kullanım alanı resim ve video işleme alanında kullanılır. Belirli bir resim içerisinde belirli bir objenin bulunduğunu algılanması gibi.

Unsupervised learning

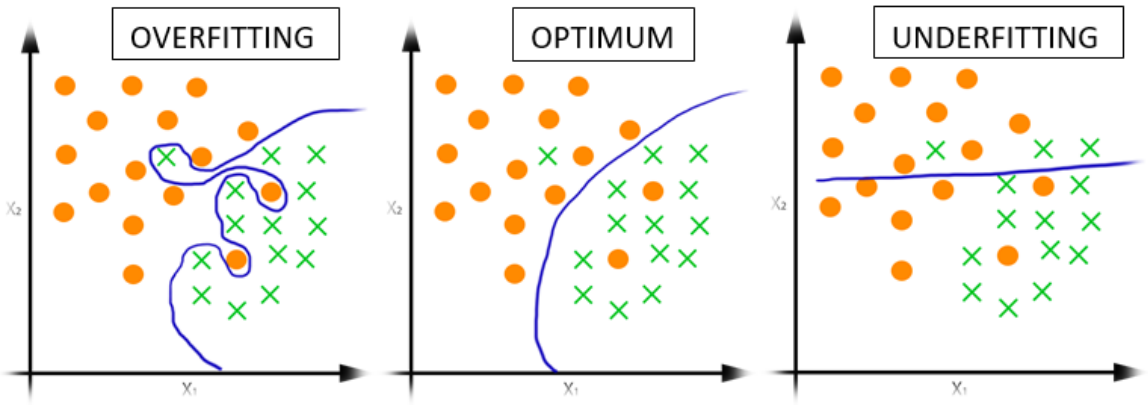
Bu eğitim yöntemi biraz daha ilginç bir şekilde çalışır. Unsupervised learning modelleri verilen veri üzerinden öğrenir, ancak bu verilerin bir çıkış veya özellikle bağlandıkları bir sonuç yoktur. Çok sayıda veri parçasını bir algoritmaya verdikten sonra bir çıkarım yapması umulur. Bu tür algoritmaların amacı verilerin altında yatan bağlantıları ortaya çıkararak veriyi daha iyi bir şekilde anlamaktır. Bunların örnekleri gruplama ve bağlantı kurmak olarak ikiye ayrılabilir. Gruplama algoritmaları verinin içindeki belirli merkezlerin etrafında toplanan verileri ortaya çıkarmaktır. Bağlantı kurma ise verilerin arasında herhangi bir bağlantı veya benzerlik olup olmadığını bulmak olarak özetlenebilir. Bu tür algoritmalar genel olarak sitelerin size öneri yapmasında, veya demografiklerin genel trendlerini ve yatkınlıklarını bulmaktır.

Overfitting

Over fitting dediğimiz olay algoritmaların eğitimi sırasında ortaya çıkabilecek bir hatadır. Bu hata algoritmanın veri üzerinde fazla detaylı bir sonuç çıkarılması ile ortaya çıkar. Bu sorun veri eksikliğinden veya aynı verinin çok fazla training için kullanılması nedeniyle oluşur. Overfitting sebebiyle ileride yapılacak tahminler çok sınırlı olacaktır. Önceki verilerle neredeyse birebir uyuşmayan veri üzerinde tahmin yapılamaz hale gelir, yapılsa bile üzerinde tahmin yapılan özellikleri yakın ise doğru sonuç elde edilemez.

Underfitting

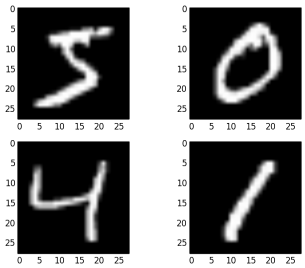
Underfitting, adından da anlaşılabileceği üzere, overfitting'in tam tersi bir sorun. Veri üzerine yeterli bir algoritma kullanmadığımız zaman ortaya çıkar. Algoritmanın çıkarımları ve sonuç oalrak da tahminleri fazla yüzeysel kalır. Underfitting ortaya çıkarsa aynı şekilde doğru bir tahmin yapamayacaktır. Yeterli tekrar eğitim yapılmaması veya etkisiz bir algoritma kullanılması ile underfitting ortaya çıkabilir.



Makine Öğrenim algoritmaları

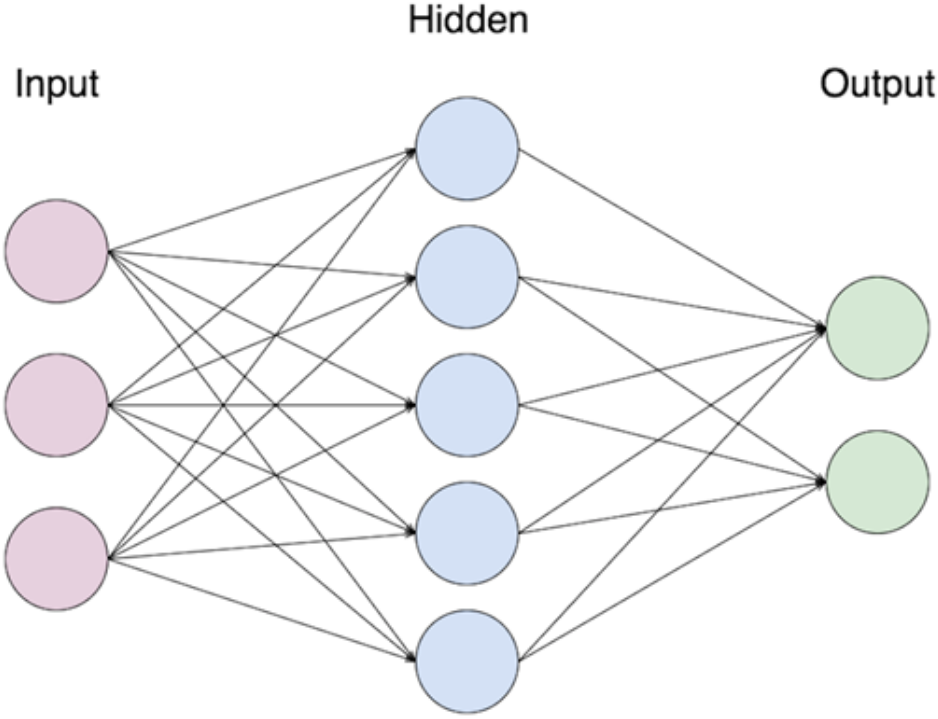
Bu kadar algoritma kelimesini kullandıktan sonra, bu algoritmaların birkaçına bakmanın zamanı geldi. Bu algoritmaların hepsi belirli matematik ve istatistik prensiplerine dayalıdır. Bu işlemlerin büyük veri setleri üzerinde ve hızlı bir şekilde yaparak makinelere belirli işleri yapmayı öğretebiliriz. Bu şekilde herşeyi insan tarafından kodlamak yerine bu programları kendi projelerimizde veya farklı uygulamalarda kullanmaya daha fazla kaynak ayrılabilir.

Neural Networks



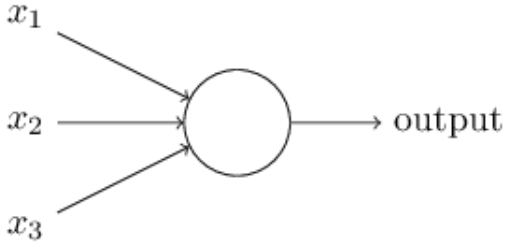
Neural Networks

Büyük ihtimalle makina öğrenimi dendiğinde ilk akla gelen sistemlerdir. Neural Networkler aslında Deep learningin yapıldığı ana alandır. Deep learninging'deki "Deep" kavramı bu tür netlerin derinliğinden gelmektedir.
Mesela yandaki resimleri ele alalım. Insanlar için hangisinin 5 veya 4 olduğunu kolayca anlayabiliriz, peki bunu bilgisayara aktarabilir miyiz? Mesela 9 için altında çizgi olan bir daire olduğunu nasıl bilgisayara anlatırız? Bir tanesini doğru olarak sınıflandırabilen bir program yazarsak, daha farklı bir el yazısını tanıyabilecek mi? Uzun bir süre bunun gibi sorunlardan dolayı resim işleme imkansıza yakındı. Son zamanlarda makina öğrenimde yapılan gelişmeler sonucunda mümkün. Basitçe konuşmak gerekirse neural networkler veri seti üzerinden belirli kurallar çıkarak resimleri sınıflandırabilir. Yani çalışma yöntemleri ise supervised learningdir. "Training set" dediğimiz eğitim için ayrılmış sonucu belirli setler kullanılarak sonuca ulaşmak üzerine kuruludur. Şimdi bu sistemin alt başlıklarına göz atalım



Perceptron

Perceptronlar neural netlerin ana tapım malzemesi oalrak düşünülebilir. Perceptronlar 1950 ve 1960 yıllarında Frank Rosenblatt tarafından geliştirilmiştir. Basitçe konuşmak gerekirse birçok değer alıp, tek bir değer dışarı veren sistemlerdir. Beyin içindeki her bir nöronun perceptron



olarak düşünebiliriz.
Mesela yukarıdaki perceptron'un üç farklı giriş verisi var. Sonrasında perceptron içinde bu verilerin ağrılıklı toplamını alıyoruz. Ağırlık terimini ileride tekrar değineceğiz. Sonuç olarak binary bir değer verecektir. Şimdiki örnek pek gerçekçi değil ama anlamamızı kolaylaştıracak.
Mesela haftasonuna girdiğimizi düşünelim.

1. Hava durumu güneşli mi?
2. Netflix çalışıyor mu
3. Dışarı çıkan arkadaş var mı?

Mesela bu bilgilerin doğru olanları 1, yanlış olanları ise 0 verdiğini varsayalım. Mesela o gün

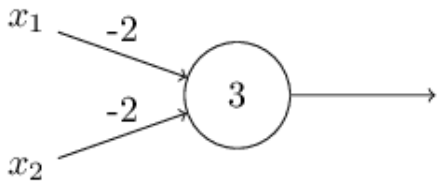
Mesela bu bilgilerin doğru olanları 1, yanlış olanları ise 0 verdiğini varsayalım. Mesela o gün tüm arkadaşlarınız sizi bi festivale davet ediyor ve hava çok güzel, ama aylardır beklediğiniz yeni bir dizi Netflix'e yeni bir dizi gelmiş. Bu örnekte x2'nin ağırlığı daha fazla olacaktır, bu sebepten dolayı içerde kalmayı seçebilirsiniz. Perceptronlar da aynı işlemi yapar. Giren verilerin ağırlıklarından bir sonuç ortaya çıkararak, bir sonuç verir. Bu ağırlıkları ve eşikleri değiştirerek farklı modeller elde edebiliriz. İşte burada istatistik ve matematik işin içine giriyor. Kullanılan bu algoritmalar aldığı yüzlerce veya binlerce veriyi, alması gerek sonuca göre değiştirerek en uygun yöntemi bulur. Tabi doğru algoritmayı kullanırsanız.

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases} \qquad (1)$$

Bunu biraz daha kolaylaştıralım. Öncelikle yukarıdaki ifadeyi dot product (bilişim) yoluyla yazacağız. Yani hem ağırlıkları hem verileri birer vector olarak işlem yapacağız. $W \cdot x \equiv \sum_j w_j x_j$ buradaki W ve X değerlerinin componentleri olarak da önceki ağırlık ve veri değerlerini alıyoruz. Sonraki adım ise eşik değerini denklemin diğer tarafına alarak yerine perceptronların bir özelliği olan "bias", yani önyargı değerini ekliyoruz. Sonuç olarak elimizde aşağıdaki gibi bir denklem ortaya çıkıyor.

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases} \qquad (2)$$

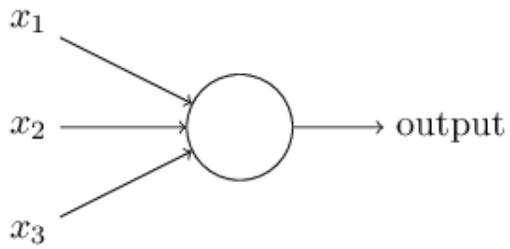
Buradaki önyargı değeri ne kadar yüksek ise, o perceptronun 1 verme olasılığının ne kadar büyük olduğuna karşılık gelir.



Yukarıda gördüğünüz şekilde perceptronların nasıl işlem yaptığının bir örneğidir. Bu perceptrona x1 = 1 ve x2 = 0 değerlerini verdiğimizde işlemimiz: (1)*-2 + (0)*-2 +3 = 1 olacaktır. Bu değerler de sonra ileriki perceptronlara iletilebilir.

Sigmoid nöronları:

Şimdi ileride bizim için önemli olacak bir nöron tipine bakalım. Mesela bir neural netinizin olduğunu düşünün, ve bunu bir tür objeyi algılamak için kullanmak istiyorsunuz. Bu öğrenimi algoritmanızın tamamlaması için ise ağırlık değerleriniz küçük değişiklikler yaparak sonucu doğruya getirmek isteriz. Bu ağırlıklardaki kü.ük değişiklikler, sonuçlardaki küçük değişikliklere göre değişecektir. Bu sayede algoritmamız öğrenebilir. Ancak bütün neural netimiz perceptronlardan oluşsaydı, bir tane ağırlıktaki küçük bir değişim bile bütün sistemin tamamen tepe taklak hale gelmesine neden olabilirdi. Yani bir resim için ayarlamış olsak bile, diğer resimlerdeki davranışlar tamamen yanlış hale gelebilirdi. Bu sorunu Sigmoid nöronlarıyla çözüyoruz.



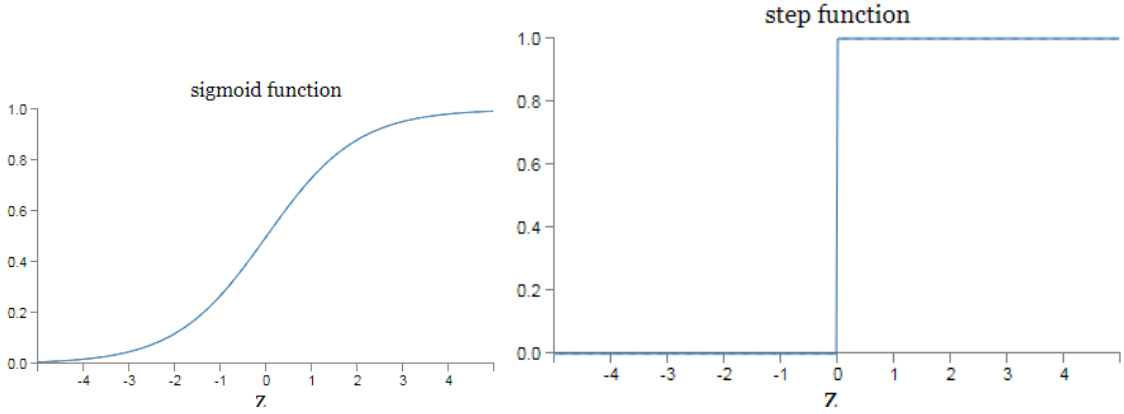
Sigmoid nöronları tam olarak aynı şekilde yapılandırılmıştır. Ancak en büyük farkımız, x değerlerimiz 0 ile 1 arasında herhangi bir değeri alabilir. Mesela x1 = 0.782 bile olabilir. Sigmoid nöronlarının da ağırlık ve önyargı değerleri bulunmaktadır, ancak formülümüz: $\sigma(w \cdot x+b)$ şeklinde yazılır. Buradaki σ işareti ise şu şekilde tanımlanır:

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}. \qquad (3)$$

Daha da uzatmak ve formülün son halini göstermek gerekirse, sonucumuz:

$$\frac{1}{1+\exp(-\sum_j w_j x_j - b)}. \qquad (4)$$

İlk bakışta sigmoid ve perceptronlar tamamen farklı olarak görülebilir, ancak aralarında birçok benzerlik bulunmakta. Mesela, z≡w·x+bz≡w·x+b değerinin büyük bir pozitif sayı olduğunu varsayalım. O zaman $e^{-z} \approx 0 e-z \approx 0$ ve formüle yerleştirdiğimizde ise: $\sigma(z) \approx 1 \sigma(z) \approx 1$ halini alır. Yani büyük pozitif bir değer aldığımızda hem sigmoid nöronlar hem de perceptronlar 1 değerini verecektir. Tam tersinde ise aynı şekilde büyük negatif bir değer iki tarafta da 0 değerini alacaktır. Bu yeni matematiksel formu grafiğe dökerek bu durumu daha iyi anlayabiliriz.

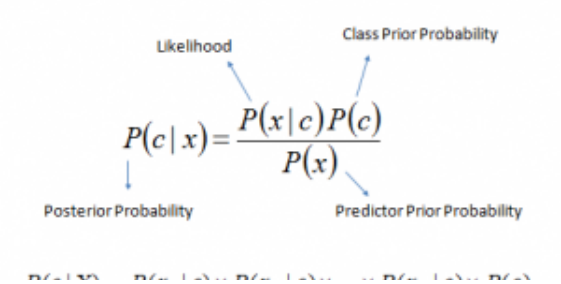


Logistic Regression

Instead of predicting *exactly* 0 or 1, **logistic regression** generates a probability—a value *between* 0 and 1, exclusive. For example, consider a logistic regression model for spam detection. If the model infers a value of 0.932 on a particular email message, it implies a 93.2% probability that the email message is spam. More precisely, it means that in the limit of *infinite* training examples, the set of examples for which the model predicts 0.932 will actually be spam 93.2% of the time and the remaining 6.8% will not.

Naive Bayes

Bu sınıflandırma yöntemi Bayes Teorisi üzerine kuruludur. Bu teorinin özelliği, belirleyici faktörler arasında bir bağlantı olmadığını varsayarak çalışmaktadır. Bunu makina öğrenim terminolojisine dökersek, sonuca ulaşmak için kullandığımız özellikler arasında bağlantı olmadığını varsayar. Yani bir meyve yaklaşık 10 cm geniş, kırmızı ve yuvarlaksa elma olabilir. Bunların her biri birbirine bağlı olsa da olmasa da, bu meyvenin elma olma olasılığına ayrı olarak katkı sağlar, buna ise "Naive" denir.



$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Artılar:

- It is easy and fast to predict class of test data set. It also perform well in multi class prediction
- Test veri setindeki sonuçları kolay ve hızlıca tahmin edebilir, aynı zamanda çoklu sonuç arasından tahmin yapıyorsak iyi bir seçim.
- Eğer özellikler gerçekten birbirlerinden bağımsız ise Naive Bayes, logistic regression gibi algoritmalardan daha iyi şekilde çalışır ve daha az veriye ihtiyaç duyar.
- Kategorisel veri ve sayısal verilerin birlikte bulunduğu setlerde iyi sonuçlar verir.

Eksiler:

- Eğer test setinde, train setinde bulunmayan bir kategori bulunuyorsa, bu kategori otomatik olarak sıfır olasılık olarak alınıp sonuçların doğruluğunu düşürür. Buna "Sıfır frekans" (Zero frequency) adı verilir. Çözüm olarak genel çözüm "Laplace Estimation" dır
- Naive Bayes aynı zamanda kötü bir tahmin algoritması olarak bilinmektedir. Bu sebeple olasılık çıktısında "predict_proba" genelde umursanmaz.
- Gerçek hayatta birbirinden tamamen kopuk verilerin bulunması çok zor, hatta imkansızdır.

Evrimsel Programlama

1. **Evrimsel programlamanın tarihçesi**

İlk olarak 1960'ta Lawrence J. Fogel tarafından ortaya atılmıştır. Fogel, bir bilgisayar araştırmacısıdır ve evrimsel programlamanın babası olarak tanınmaktadır. 1965'ten 2007'ye kadar evrimsel programlamanın prensiplerini endüstri, ilaç gibi alanlar üzerinde kullanmış ML ve AI alanlarında konferanslar düzenlemiştir.
UCLA'da "On The Origin of Intellect" başlığıyla doktorasını yapmıştır. His Evrimsel programlamayı ön plana çıkaran en büyük eseri, Owens ve Walsh ile beraber yazdığı "Artificial Intelligence Through Simulated Evolution" kitabıdır.

2. **Seleksiyon**

Evrimsel programlamayı tanımlamadan önce birtakım kavramlardan bahsetmemiz gerekiyor. İlk kavramımız seleksiyon.
Belirlenen bir alanda yaşayan canlılar ve türler kendilerini kopyalayabiliyorlar ise yaşamaya devam ederler. Yapamazlarsa ölürler. Kopyalama derken o ortamın koşullarında yaşamaya en dayanıklı ve en uygun canlılar ve türler yaşayacaktır. Uyumlu olmayanların nesilleri zamanla tükenecektir. Tabii ki bu, başlangıçta belirlediğimiz canlı havuzunda çeşitliliğin olduğunu varsayarak çalışır.

Seleksiyon, veya doğal seçilim, bireylerin "problem çözmeye" sevk edildikleri ortamlarda performans göstermeleriyle gerçekleşir. Hayatta kalabilmeleri için, üreyebilene kadar hayatta kalmaları gerekir.

3. **Evrim**

Kopyalamadan bahsettik. Ancak bu evrim sayılmaz. Peki kendini kopyalayabilen bir canlı türü nasıl evrimleşebilir?

a. **Mutasyon**

Bir yol mutasyon. Tekrarlanabilir bir tür çeşitliliktir. Ancak tehlikeli ve rastgeledir. Sonuç olarak etkili ama verimli olmayan bir evrim türüdür.

b. **Cross-over**

d. **Cross-over**

Sağlıklı genlerin etkilemek yerine var olan genler paylaşılırsa daha kolay olur. Bu, en basit haliyle üremedir.

Canlı kendini kopyaladığı zaman bir çeşitlilik oluşmaz, ancak üremeyle beraber genler çaprazlanır ve genlerde çeşitlilik oluşur. Mutasyondan çok daha güvenli ve verimlidir. Çok daha az da rastgeledir. Ancak tekrarlanabilir çeşitlilik sağlamaz. Bütün kombinasyonlar kullanıldığı zaman farklı bir çeşitlilik olmayacaktır. Yani çeşitliliği devam ettirebilmek için mutasyona yine de bir ihtiyaç vardır.

4. Evrimsel Programlama Nedir?

Yukarıda bahsettiğimiz kavramlardan yola çıkarak diyebiliriz ki seleksiyon da evrim de genetik bir probleme çözüm değildir. Canlıları evimleştirerek probleme bir çözüm arar ancak bir çözüm sunmaz.

Evrimsel programlama da buna benzerdir. Var olan bir problemin çözümü değil de, o probleme çözüm aramaya kullanılacak bir metodolojidir.

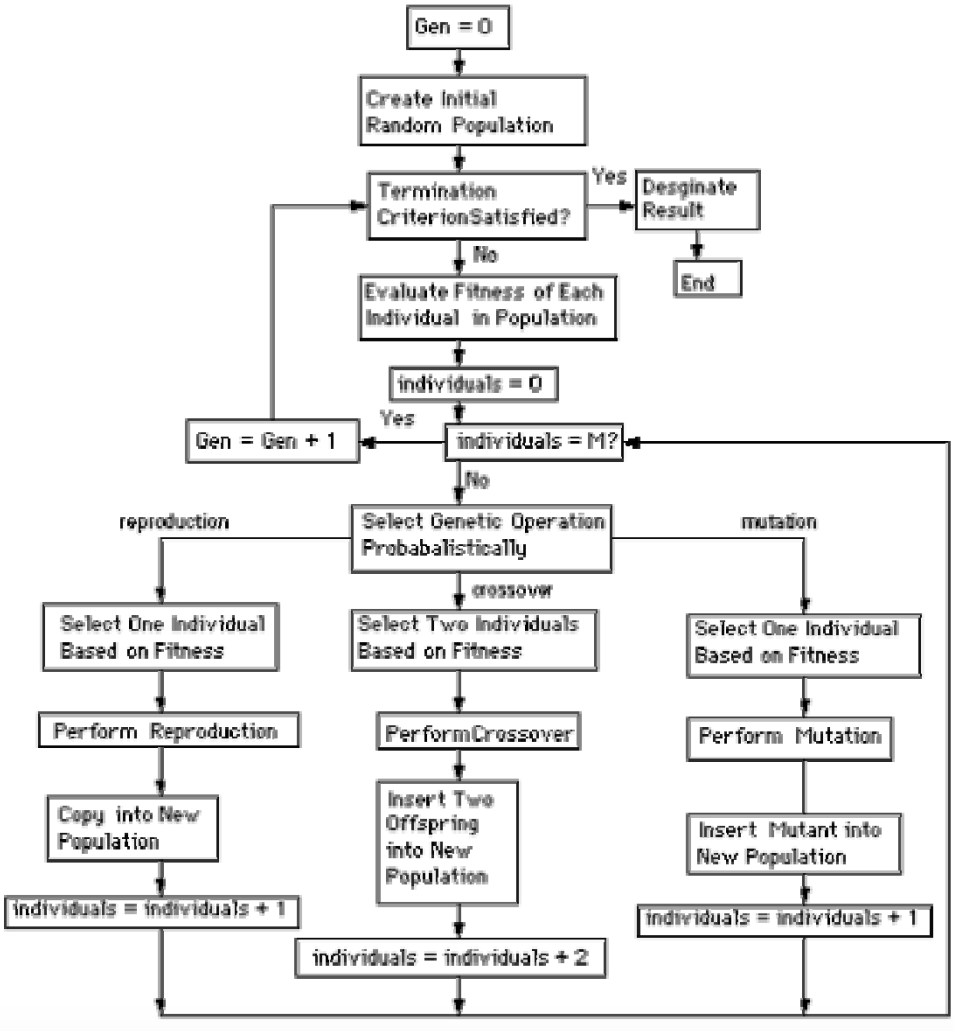
Başka bir deyişle evrimsel programlama, genetik algoritmalara benzer bir süreçten geçen, ancak var olan organizmanın direkt olarak kopyalanması yerine "parent" ile "offspring" arasında davranışsal bağlantılara vurgu yapan bir optimizasyon tekniğidir.

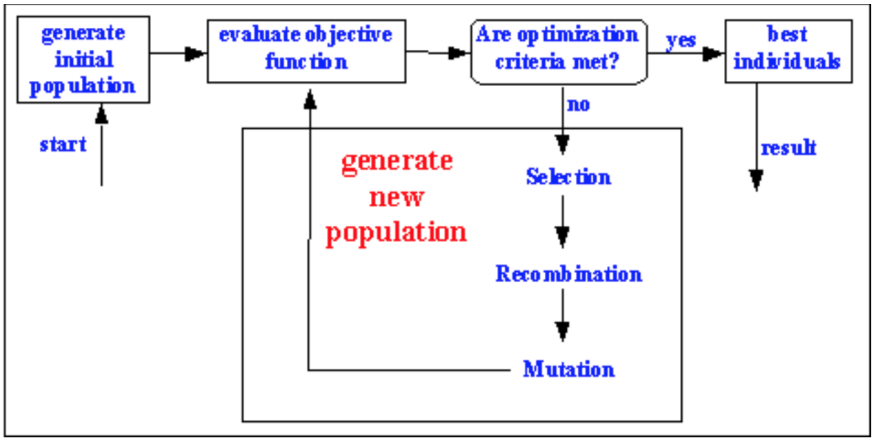
5. Evrimsel Programlamanın Aşamaları

a. Rastgele bir şekilde başlangıç çözümleri için canlı popülasyonu seçilir. Çözümlerin sayısı optimizasyon için önemlidir, ancak 1'den büyük koşulu hariç uygun ve uygun olmayan çözümlerin sayıları bağlamında kesin bir cevap yoktur.

b. Bütün çözümler yeni bir popülasyona kopyalanır. Bütün "offspring" çözümleri mutasyon aracılığıyla evrime uğratılırlar. Mutasyonlar, küçükten ekstreme kadar bir çeşitlilik içerir.

c. Her "offspring" çözüm evrim sonucu o ortamda hayatta kalabilmeye olan kabiliyetleri, "fitness"ları ölçülür.





6. **Sudo Kod**

```
 Algorithm EP is

// start with an initial time
t := 0;

// initialize a usually random population of individuals
initpopulation P (t);

// evaluate fitness of all initial individuals of population
evaluate P (t);

// test for termination criterion (time, fitness, etc.)
while not done do

     // perturb the whole population stochastically
     P'(t) := mutate P (t);

     // evaluate it's new fitness
     evaluate P' (t);

     // stochastically select the survivors from actual fitness
     P(t+1) := survive P(t),P'(t);

     // increase the time counter
     t := t + 1;
od
 end EP.
```

Selim Bilgin

3. Uninformed Search (Blind Search)

Nedir?

Uninformed Search, elinde domain-specific knowledge olmadan bir search tree yaratır. Genelde brute force algoritmaları gerektirir. Bir sonuç çıkana kadar karşısına çıkan bütün nodları dener ve bir sonuca ulaşır.

Ne zaman kullanılır?

Bir oyun oynadığınızı var sayın (satranç, COD, LOL, YGO). Bu oyunlarda her an yapabileceğiniz birçok hamle vardır. Her hamle farklı sonuçlar doğurur ve bu sonuçlar kazanıp kazanmadığınızı belirler. Uninformed Search algoritmaları yapılabilecek hamleleri analiz eder ve sonuca giden yolları bulur. Bu yöntem elinizdeki verilerin bir hipoteze uyup uymadığını test etmek için de kullanılabilir. Neticede bu yöntem, olabilecek senaryoları deneyerek sizi hedefe (goal state) ulaştıran bir yol yaratır.

Çeşitleri

Breadth-First Search (BFS)

BFS kök noddan komşu nodlara doğru bir search tree analiz eder. Her seviyedeki komşu nodlar analiz edildikten sonra alt seviyeye geçer. Komşuları soldan sağa analiz eden bu sistem, en alt noda ulaşana kadar tekrar tekrar başa dönerek (her seviye bittikten sonra) bütün nodları analiz eder.

Depth-First Search (DFS)

DFS kök noddan başlayarak yukarıdan aşağıya doğru nodları analiz eder. Her nod serisi bittikten sonra yukarı çıkıp bir yandaki nod serisine geçer (soldan sağa).

Depth Limited Search (DLS)

DFS gibidir ama ama nodları önceden belirlenen bir seviyey kadar analiz eder. Belirlenen seviyenin altındaki olan nodları yokmuş gibi var sayar.

Bidirectional Search

Aynı anda 2 farklı yönden (baştan sona ve sondan başa) arama yapar. 2 arama bir nodda kesişince aramayı bitirir. Bu bir brute force algoritmasıdır ve daha hızlı olmasıyla beraber daha az hafıza gerektirir.

Uniform-Cost Search

BFS gibidir ama baştan sona en az ağırlığa sahip olan nodlardan geçmeyi dener. Hedefe ulaşabilecek birçok uzun yol olabileceğinden bütün yolları denemesi gerekir. Baştan sona en az masraflı yolları seçtiği için daha ucuzdur.

IDDFS

IDDFS, BFS gibidir ama daha az hafıza gerektirir. BFS ile DFS in bir karışımı olarak verileri analiz eder.

Expert-Systems (Ivan Bratko)

Nedir?

Expert Systemlar belilrli bir konuda uzman gibi davranan sistemlerdir.

Örnekler:

1) Gelecekteki uçuşlar için ekonomi ve business koltukları arasında ayrım yapmak
2) Baskı cihazlarında kağıt problemleri analiz etmek
3) Petrol veya maden aramaları sürdürmek
4) Tıp konularında teşhis koymak

Özellikleri:

1) Uzman olunan konuda problem çözmek
2) Problem çözülmeden önce ve sonra kullanıcı ile iletişim kurmak
3) Domain Knowledge dan faydalanmak
4) Sonuçları kullanıcıya açıklamak
5) Bilgiyi If ve Then üzerine kurulmuş kurallar ile analiz eder
6) Baştan sona ve sondan başa analiz yapabilir

Standart yapısı

1) Knowledge Base: Kuralları bağlı bir Expert System'da bilgiler kurallar olarak bulunur. Her kural bir If ve Then bağlantısı içerir ve harekete geçirilince bir iş yapar.

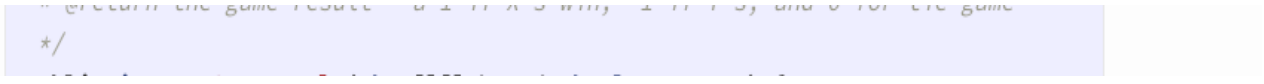
2) Database: If içeren functionların değerlendirmesi için gerekli olan bilgiye sahiptir

3) Inference engine: System bir çözüme ulaşırken gerekli olan mantığı yürütür. Databasedeki bilgilerle Knowledge Basedeki bilgileri bağlar.

4) Explanation Facilities: Kullanıcının Expert System'a bir karara nasıl vardığı hakkında sorduğu sorulara yanıt verir.

5) User Interface: Soruyu soran kullanıcı ile Expet system'ı bağlar.

Uninformed Search Örneği (Tic tac Toe)



```java
public int getGameResult(char[][] board, boolean xTurn) {

        // If the game is already over with this board, return the result

        if (gameOver(board))
                return gameResult(board);

        // If the game is still going, check all the possible moves
        // and choose the one with the most favorable outcome for the player

        int result = xTurn ? -1:1;

        for (int i = 0; i < board.length; i++)
                for (int a = 0; a < board[i].length; a++) {
                        if (board[i][a] != ' ')
                                continue;
                        // Place the move, then run the function recursively,
                        // then undo the move
                        board[i][a] = xTurn ? 'X':'O';
                        int tempResult = getGameResult(board, !xTurn);
                        board[i][a] = ' ';

                        // Check if the result is favorable for the player
                        if ((xTurn == tempResult > result) ||
                                (!xTurn && tempResult < result))
                                result = tempResult;
                }

        return result;
}
```

Bu Kodun Özellikleri

1. **Oyun bitti mi bakar**— Bütün kutular dolu ise veya bir oyuncu kazandıysa sonucu verir
2. **Bütün karelere bakar**
    - Bir kutu dolu ise sonraki kutuya geçer
    - Oyuncunun rengine bağlı olarak “x” veya “o” işareti yerleştirir
    - Dalı update ederek oyuncu için ideal seneryoyu bulmayı dener

Bu kodun açıklaması

İdeal bir şekilde oynandığı var sayılınca bir oyunu kimin kazanacağını veren bir koddur. Bu tarz bir algoritma aslında bir tree yaratmadığı için daha kolaydır. Bu kod yukarıdan aşağıya doğru bütün nodları geri dönmeden kontrol ettiğinden aslında bir DFS örneğidir. Genel olarak başa dönen bir yapıya sahip olduğundan *recursive* bir algoritmadır.