
buzdağı

Yayın Numarası: 2406

1. Baskı: Nisan 2024

ISBN 978-625-98552-3-3

Sertifika Numarası: 41825
Buzdağı Yayınevi
Yeni Bağlıca Mah.
1068 Sok. No:4/1
Etimesgut/ANKARA
t: +90 312 219 55 43
info@buzdagiyayinevi.com

Baskı
Salmat Basım Yayıncılık Ambalaj
San. ve Tic. Ltd. Şti.
47771

MAKİNE ÖĞRENMESİ SİSTEMLERİ TASARLAMAK

Üretime-Hazır Uygulamalar için Yinelemeli Bir Süreç

Chip Huyen

Genel Yayın Yönetmeni
Fatih ÖZDEMİR

Yazar
Chip HUYEN

Çevirmen
Özgür KAYA

Editör
Dr. Mustafa Murat ARAT

Son Okuma
Muhammed TOPRAK

Dizgi
Veysel TOPRAK

İçindekiler

bulunduğunda, fiyat optimizasyon servisi bu yolculuk için optimal fiyatı tahmin etmek üzere tahmin edilen yolculuk sayısını ve tahmin edilen sürücü sayısını ister.[22]

Verileri ağlar üzerinden göndermek için kullanılan en popüler istek stilleri REST (representational state transfer) (ÇN: Temsili durum transferi) ve RPC'dir (remote procedure call) (ÇN: Uzaktan yordam çağrısı). Bu stillerin ayrıntılı analizi bu kitabın kapsamı dışındadır fakat aralarındaki başlıca fark, REST ağlar üzerinden yapılan istekler için tasarlanmışken, RPC'nin "uzak bir ağ servisine yapılan bir isteği, programlama dilinizdeki bir fonksiyonu veya metodu çağırmanın aynısına benzetmeye çalışmasıdır." Bu nedenden ötürü "REST, herkesin kullanımına açık genel API'ler için baskın olan stil gibi görünmektedir." RPC yazılım çerçevelerinin ana odak noktası, genellikle aynı veri merkezi içindeki aynı organizasyona ait servisler arasındaki isteklerdir.[23]

Bir REST mimarisinin uygulanmasının RESTful olduğu söylenir. Birçok kişi REST'i HTTP olarak düşünse de REST tam olarak HTTP anlamına gelmez çünkü HTTP sadece REST'in bir uygulamasıdır.[24]

3.5.3 Gerçek-Zamanlı Taşıma Aracılığıyla Veri Gönderimi

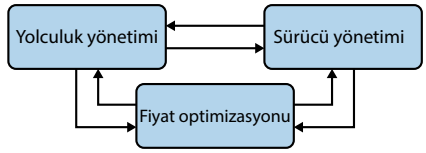
Gerçek-zamanlı taşımaların arkasındaki motivasyonu anlamak için sürücü yönetimi, yolculuk yönetimi ve fiyat optimizasyonu şeklinde üç basit servisi olan araç paylaşımı örneğine geri gidelim. Bir önceki bölümde fiyat optimizasyon servisinin her bir yolculuk için optimal fiyatı tahmin etmek amacıyla yolculuk ve sürücü yönetimi servislerinden gelecek verilere nasıl ihtiyaç duyduğundan bahsettik.

Şimdi, sürücü yönetimi servisinin kaç tane sürücüyü harekete geçireceğini bilmek için yolculuk yönetimi servisinden gelecek yolculuk sayısını da bilmesi gerektiğini hayal ediniz. Ayrıca, sürücü yönetimi servisi, potansiyel sürücüler için teşvik olarak kullanmak üzere fiyat optimizasyonu servisinden tahmin edilen fiyatları bilmek ister (örneğin, şimdi yola çıkarsanız 2 kat fazla ücret alabilirsiniz). Benzer şekilde, yolculuk yönetimi servisi, sürücü yönetimi ve fiyat optimizasyon servislerinden veri isteyebilir. Verileri önceki kısımda tartışıldığı gibi servisler aracılığıyla gönderirsek, Şekil 3.8'de gösterildiği gibi bu servislerin her birinin diğer iki servise istek göndermesi gerekir.

[22]Uygulamada, fiyat optimizasyonu, her fiyat tahmini yapması gerektiğinde, tahmin edilen yolculuk/sürücü sayısını istemek zorunda kalmayabilir. Önbelleğe alınan tahmin edilmiş yolculuk/sürücü sayısını kullanmak ve her dakikada bir yeni tahminler istemek yaygın bir uygulamadır.

[23]Kleppmann, Designing Data-Intensive Applications.

[24]Tyson Trautmann, "Debunking the Myths of RPC and REST," Ethereal Bits, 4 Aralık 2012 (Internet Archive ile erişildi), *https://oreil.ly/4sUrL*.

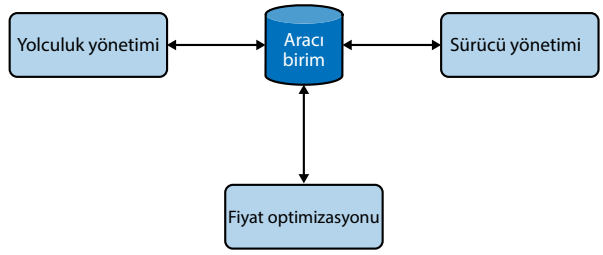


Şekil 3.8: *İstek-güdümlü mimaride her bir servisin diğer iki servise istekler göndermesi gerekir.*

Sadece üç servisle bile veri gönderme şimdiden karmaşık hâle gelmektedir. Belli başlı internet şirketlerinin sahip olduğu gibi binlerce servis olmasa da yüzlerce servise sahip olduğunuzu düşününüz. Servisler arası veri aktarımı patlayabilir ve bir darboğaza dönüşerek tüm sistemi yavaşlatabilir.

İstek-güdümlü veri gönderme senkronize bir iştir: Hedef servis, isteğin iletilebilmesi için isteği dinlemek zorundadır. Eğer fiyat optimizasyon servisi, sürücü yönetimi servisinden veri isterse ve sürücü yönetimi servisi çalışır durumda değilse fiyat optimizasyon servisi, istek zaman aşımına uğrayana kadar isteği yeniden göndermeye devam eder. Ve eğer fiyat optimizasyon servisi bir yanıt almadan önce çalışamaz hâle gelirse, bu yanıt kaybolacaktır. Çalışmayan bir servis, kendisinden veri gerektiren tüm servislerin çalışmamasına neden olabilir.

Peki, hizmetler arasında veri göndermeyi koordine eden bir aracı birim (ÇN: Broker) olursa ne olur? Servislerin doğrudan doğruya birbirinden veri istemesi ve karmaşık bir servisler arası veri gönderme ağı oluşturması yerine, her bir servisin Şekil 3.9'daki gibi sadece aracı birim ile iletişim kurması gerekir. Örneğin, diğer servislerin, sürücü yönetimi servislerinden bir sonraki dakika için tahmin edilen sürücü sayısını istemesi yerine her ne zaman sürücü yönetimi servisi bir tahminde bulunsa ve bu tahmin bir aracı birime yayımlansa ne olur? Hangi servis sürücü yönetimi servisinden veri isterse, o aracıyı en son tahmin edilen sürücü sayısı için kontrol edebilir. Benzer şekilde, fiyat optimizasyon servisi bir sonraki dakika için dalgalanma ücreti hakkında bir tahmin yaptığında, bu tahmin aracı birime yayımlanır.



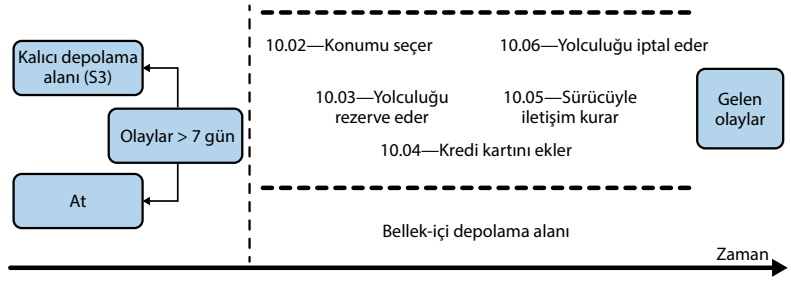
Şekil 3.9: *Bir aracı birim kullanıldığında, bir servisin diğer servislerle iletişim kurmak yerine sadece bu aracı birimle iletişim kurması gerekir.*

Teknik olarak bir veritabanı bir aracı birim olabilir. Yani her bir servis bir veritabanına veri yazabilir ve veriye ihtiyaç duyan öteki servisler bu veritabanını okuyabilir. Bununla birlikte 77. sayfadaki "Veritabanları Aracılığıyla Veri Gönderimi" kısmında da bahsedildiği gibi veritabanlarını okumak ve veritabanlarına yazmak, katı gecikme gereksinimleri olan uygulamalar için çok yavaş bir iştir. Verilere aracılık etmek için veritabanları kullanmak yerine bellek-içi depolamayı kullanırız. Gerçek-zamanlı taşımalar, servisler arasında veri aktarımı için bellek-içi depolama gibi düşünülebilir.

Bir gerçek-zamanlı taşımaya yayımlanan bir veri parçasına bir olay (ÇN: Event) denir. Bu mimari bu nedenle olay-güdümlü olarak da adlandırılır. Bir gerçek-zamanlı taşıma bazen "olay veri yolu" (ÇN: Event bus) olarak da isimlendirilir.

İstek-güdümlü mimari, verilerden ziyade mantığa dayanan sistemler için iyi çalışır. Olay-güdümlü mimari, veri-ağırlıklı sistemler için daha iyi çalışır.

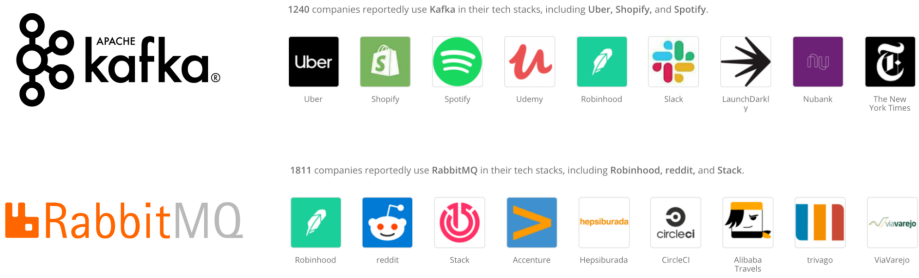
Gerçek-zamanlı taşımaların en yaygın iki örneği yayınlama-abonelik (ÇN: Publish-subscribe) kısaltması olan pubsub ve mesaj kuyruğudur. pubsub modelinde herhangi bir servis bir gerçek-zamanlı taşımada farklı konulara yayın yapabilir ve bir konuya abone olan herhangi bir servis o konudaki tüm olayları okuyabilir. Veri üreten servisler, verilerini hangi servislerin tükettiğiyle ilgilenmez. Pubsub çözümlerinin genellikle bir saklama politikası vardır. Yani veriler, silinmeden veya kalıcı bir depolamaya (Amazon S3 gibi) taşınmadan önce belirli bir süre (örneğin, yedi gün) gerçek-zamanlı taşımada tutulur. Bakınız Şekil 3.10.



Şekil 3.10: *Gelen olaylar atılmadan veya daha kalıcı bir depolama alanına taşınmadan önce bellek-içi depolama alanında depolanır.*

Bir mesaj kuyruğu modelinde, bir olay, planlanmış tüketicilere sahiptir (planlanmış tüketicilere sahip bir olay, mesaj olarak adlandırılır) ve mesaj kuyruğu mesajı doğru tüketicilere ulaştırmakla sorumludur.

Pubsub çözümlerinin örnekleri Apache Kafka ve Amazon Kinesis'tir.[25] Mesaj kuyruklarının örnekleri Apache RocketMQ ve RabbitMQ'dur. Her iki paradigma da son birkaç yılda çok ilgi görmüştür. Şekil 3.11, Apache Kafka ve RabbitMQ kullanan bazı şirketleri göstermektedir.



Şekil 3.11: *Apache Kafka ve RabbitMQ kullanan şirketler.*
Kaynak: Stackshare'den (https://oreil.ly/OqAgL) bir ekran görüntüsü.

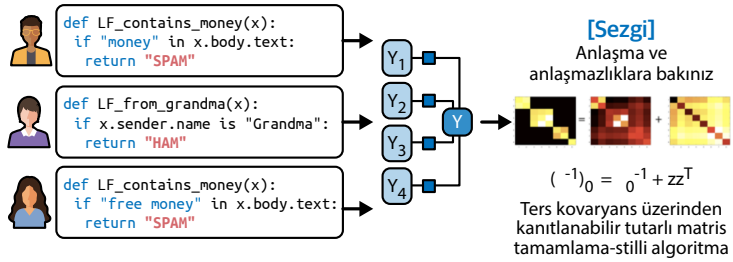
3.6 Akış İşlemeye Karşı Yığın İşleme

Verileriniz veritabanları, veri gölleri veya veri ambarları gibi veri depolama motorlarına ulaştıktan sonra tarihi verilere dönüşür. Bu, akışlı verilerin (hâlen akış hâlinde olan verilerin) tam tersidir. Tarihi veriler genellikle yığın işlerde, yani periyodik olarak başlatılan işlerde işlenir. Örneğin, günde bir kez, son gündeki tüm yolculuklar için ortalama dalgalanma ücretini hesaplamak için bir yığın işi çalıştırmak isteyebilirsiniz.

Veriler yığın işlerde işlendiğinde buna yığın işleme diyoruz. Yığın işleme, onlarca yıldır araştırma konusu olmuştur ve şirketler yığın verileri verimli bir şekilde işlemek için MapReduce ve Spark gibi dağıtılmış sistemler geliştirmişlerdir.

Apache Kafka ve Amazon Kinesis gibi gerçek-zamanlı taşımalarda verileriniz olduğunda, akışlı verileriniz olduğunu söyleriz. Akış işleme, akışlı veriler üzerinde hesaplama yapmak anlamına gelir. Akışlı veriler üzerinde gerçekleşen hesaplamalar da periyodik olarak başlatılabilir ancak periyotlar genellikle yığın işler için olan periyotlardan çok daha kısadır (örneğin, her gün yerine her beş dakikada bir). Akışlı veriler ilgili hesaplamalar, ihtiyaç duyulduğunda da başlatılabilir. Örneğin, bir kullanıcı ne zaman bir yolculuk talep ederse şu anda hangi sürücülerin mevcut olduğunu görmek için veri akışınızı işlersiniz.

[25] Apache Kafka'nın nasıl çalıştığı hakkında daha çok şey öğrenmek istiyorsanız, Mitch Seymour'un konuyu su samurları kullanarak anlattığı çok güzel bir animasyonu (*https://oreil.ly/kBZzU*) mevcuttur!



Şekil 4.4: *Etiketleme fonksiyonlarının nasıl birleştirildiğine üst düzey bir bakış. Kaynak: Rartner vd.'nin bir görselinden uyarlanmıştır.*[12]

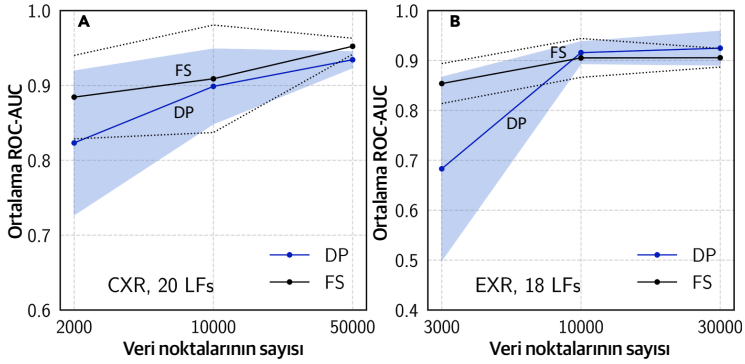
Verileriniz katı gizlilik gereksinimlerine sahip olduğunda, zayıf denetim özellikle kullanışlı olabilir. LF'ler yazmak için sadece küçük ve temizlenmiş bir veri alt kümesi gerekir. Bu LF'ler, kimse verilere bakmadan verilerin geri kalan kısmına uygulanabilir.

LF'ler kullanıldığında, konu uzmanlığı (ÇN: Subject matter expertise) versiyonlanabilir, yeniden kullanılabilir ve paylaşılabilir. Bir ekibin sahip olduğu uzmanlık bir başka ekip tarafından kodlanıp, kullanılabilir. Verileriniz veya gereksinimleriniz değişirse yeni LF'lerinizi veri örneklerinize yeniden uygulayabilirsiniz. Verileriniz için etiket üretmek amacıyla LF'leri kullanma yaklaşımı programatik etiketleme olarak da bilinir. Tablo 4.3 programatik etiketlemenin el ile verilen etiketlere göre bazı avantajlarını göstermektedir.

Tablo 4.3: *Programatik etiketlemenin el ile verilen etiketlere göre avantajları*

El ile etiket verme	**Programatik etiketleme**
Pahalı: Özellikle konu uzmanlığı gerektiğinde	Maliyetten tasarruf: Uzmanlık bir organizasyon içerisinde versiyonlanabilir, paylaşılabilir ve yeniden kullanılabilir.
Gizliliğin eksikliği: Verilerin insan etiket atayıcılara gönderilmesi gerekir	Gizlilik: Temizlenmiş bir alt örneklem kullanarak LF'ler üret ve daha sonra bu LF'leri tek tek veri örneklerine bakmadan tüm verilere uygula
Yavaş: Gereken süre, gereken etiketlerin sayısıyla doğrusal olarak ölçeklenir	Hızlı: 1.000 veri örneğinden 1 milyon veri örneğine kolaylıkla ölçeklenir
Uyarlanamaz: Her değişiklik verilerin yeniden etiketlenmesini gerektirir	Uyarlanır: Değişiklikler olduğunda, sadece LF'leri yeniden uygula!

İşte zayıf denetimin uygulamada ne kadar iyi çalıştığını gösteren bir vaka çalışması. Stanford Medicine[13] ile birlikte yapılan bir çalışmada, tek bir radyolog tarafından sekiz saatlik bir LF yazma işleminden sonra elde edilen zayıf denetimli etiketlerle eğitilmiş modeller, Şekil 4.5'te de gösterildiği gibi, neredeyse bir yıllık elle etiketleme yoluyla elde edilen verilerle eğitilen modeller ile karşılaştırılabilir performans göstermiştir. Deneyin sonuçlarıyla ilgili iki ilginç gerçek mevcuttur. Birincisi, daha fazla etiketleme fonksiyonu olmadan bile modeller daha fazla etiketlenmemiş veriyle birlikte iyileşmeye devam etmiştir. İkincisi, etiketleme fonksiyonları görevler arasında tekrar kullanılmaktaydı. Araştırmacılar, CXR (göğüs röntgenleri) (ÇN: Chest X-rays kısaltması) görevi ile EXR (ekstremite röntgenleri) (ÇN: Extremity X-rays kısaltması) görevi arasında altı farklı etiketleme fonksiyonunu tekrar kullanabildi.[14]



Şekil 4.5: *Tamamen denetimli (ÇN: Fully supervised) etiketler üzerinde eğitilen bir model (FS) ile programatik etiketler ile eğitilen bir modelin (DP), CXR ve EXR görevleri üzerinde karşılaştırılması. Kaynak: Dummon vd.*[15]

Öğrencilerim sıklıkla buluşsal yöntemler verileri etiketlemek için bu kadar iyi çalışıyorsa neden MÖ modellerine ihtiyacımız olduğunu soruyorlar. Bunun bir nedeni, etiketleme fonksiyonlarının tüm veri örneklerini kapsamayabilmesidir; bu nedenle, etiketleme fonksiyonlarıyla programlı bir şekilde etiketlenmiş veriler üzerinde makine öğrenmesi modellerini eğitebilir ve bu eğitilmiş modeli, herhangi bir etiketleme fonksiyonunun kapsamadığı veri örnekleri için tahminler oluşturmak üzere kullanabiliriz.

Zayıf denetim basit fakat güçlü bir paradigmadır. Ancak zayıf denetim mükemmel değildir. Bazı durumlarda, zayıf denetimle elde edilen etiketler kullanışlı

[13] Jared A. Dunnmon, Alexander J. Ratner, Khaled Saab, Matthew P. Lungren, Daniel L. Rubin ve Christopher Ré, "Cross-Modal Data Programming Enables Rapid Medical Machine Learning," Patterns 1, no. 2 (2020): 100019, *https://oreil.ly/nKt8E*.

[14] Bu çalışmadaki iki görev sırasıyla sadece 18 ve 20 LF kullanmaktadır. Uygulamada, her bir görev için yüzlerce LF kullanan ekipler gördüm.

[15] Dummon vd., "Cross-Modal Data Programming."

hatalara neden olabilirler. Bir keresinde, yardımcı olduğum projelerden birinde, uygulamanın ön ucu (ÇN: Frontend) artık kullanıcılara yaşlarını sormadığından, bu nedenle yaş değerleri kayıp olduğundan ve model bunları 0 ile doldurduğundan, modelin çöp çıktılar verdiğini keşfettik. Çünkü model, eğitim esnasında 0 yaş değerini hiç görmemişti, bu nedenle makul tahminler yapamıyordu.

Genel olarak, kayıp gözlemleri olası değerlerle doldurmaktan kaçınmak istersiniz, örneğin çocuk sayısının kayıp gözlemlerini 0 ile doldurmak. 0, çocuk sayısı için olası bir değerdir. Bilgileri kayıp olan kişiler ile çocuğu olmayan kişiler arasında ayrım yapmayı zorlaştırır.

Belirli bir veri kümesi için kayıp gözlemleri işlemek adına aynı anda veya ardışık bir şekilde birden fazla teknik kullanılabilir. Hangi teknikleri kullanırsanız kullanın, kesin olan bir şey var: kayıp değerleri ele almanın mükemmel bir yolu yoktur. Silme işlemiyle, önemli bilgileri kaybetme veya yanlılıkları öne çıkarma riskiyle karşı karşıya kalırsınız. İmpütasyon ile verilerinize kendi yanlılıklarınızı enjekte etme ve verilerinize gürültü ekleme veya daha kötüsü veri sızıntısı riskiyle karşı karşıya kalırsınız. Veri sızıntısının ne olduğunu bilmiyorsanız, paniklemeyiniz, 145. sayfadaki “Veri Sızıntısı” kısmında ele alacağız.

5.2.2 Ölçekleme

Bir kişinin gelecek 12 ay içerisinde bir ev satın alıp almayacağını tahmin etme görevini ve Tablo 5-2'de gösterilen verileri düşününüz. Verilerimizde Yaş değişkeninin değerleri 20 ila 40 arasındayken Yıllık Gelir değişkeninin değerleri 10.000 ila 150.000 arasındadır. Bu iki değişkeni bir MÖ modeline girdi olarak verdiğimizde, MÖ modeli 150.000 ve 40'ın farklı şeyleri temsil ettiğini anlamayacaktır. Model, her ikisini de sayı olarak görecek ve 150.000 sayısı 40'tan çok daha büyük olduğundan hangi değişkenin tahmin oluşturmak için gerçekten daha yararlı olduğuna bakmaksızın ona daha fazla önem verebilir.

Öznitelikleri modellere girdi olarak beslemeden önce, bu öznitelikleri benzer aralıklara sahip olacak şekilde ölçeklemek önemlidir. Bu süreç öznitelik ölçekleme olarak adlandırılır. Bu, yapabileceğiniz en basit şeylerden biridir ve sıklıkla modelinizde bir performans artışıyla sonuçlanır. Verilerinizi ölçeklemeyi ihmal etmek, modelinizin, özellikle gradyan-hızlandırılmış ağaçlar ve lojistik regresyon gibi klasik algoritmalarla anlamsız tahminler yapmasına neden olabilir.[4]

Özniteliklerinizi ölçeklemenin sezgisel bir yolu bu özniteliklerin $[0, 1]$ aralığında olmalarını sağlamaktır. Bir x değişkeni verildiğinde, bu değişkenin değerleri bu aralıkta olacak şekilde aşağıdaki formül kullanılarak yeniden ölçeklenebilir:

[4] Öznitelik ölçekleme bir keresinde modelimin performansını neredeyse %10 arttırdı.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

x'in maksimum değer olması durumunda, x' ölçekli değerinin 1 olacağını doğrulayabilirsiniz. x minimum değer ise ölçeklenmiş değer x' 0 olacaktır.

Özniteliğinizin keyfi bir $[a, b]$ aralığında olmasını isterseniz –deneysel olarak, $[-1, 1]$ aralığının $[0, 1]$ aralığından daha iyi çalıştığını görüyorum– aşağıdaki formülü kullanabilirsiniz:

$$x' = a + \frac{(x - \min(x))(b - a)}{\max(x) - \min(x)}$$

Keyfi bir aralığa ölçeklendirme, değişkenleriniz hakkında herhangi bir varsayımda bulunmak istemediğinizde işe yarar. Değişkenlerinizin normal dağılıma sahip olabileceğini düşünüyorsanız, onları sıfır ortalama ve birim varyansa sahip olacak şekilde normalleştirmek yararlı olabilir. Bu işlem standartlaştırma olarak adlandırılır:

$$x' = \frac{x - \bar{x}}{\sigma}$$

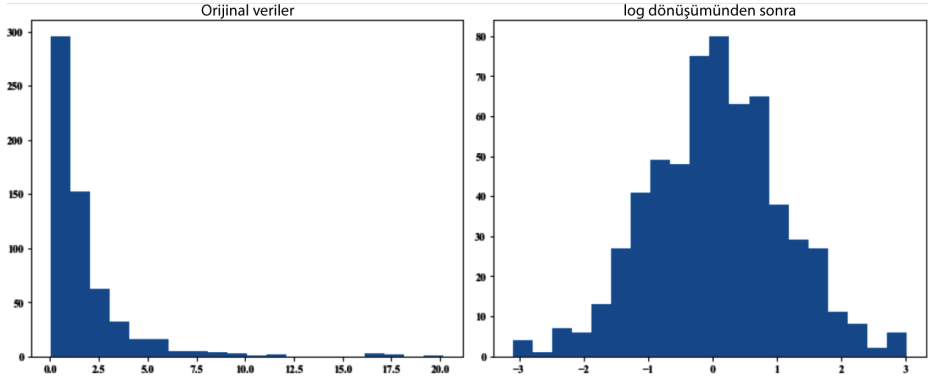
Burada $\bar{x}$, x değişkeninin ortalaması, σ ise x'in standart sapmasıdır.

Makine öğrenmesi modelleri, uygulamada çarpık dağılıma sahip özniteliklerle mücadele etme eğilimindedir. Çarpıklığı azaltmaya yardımcı olmak için yaygın olarak kullanılan bir teknik, log dönüşümüdür (*https://oreil.ly/RMwEy*): Özniteliğinize log fonksiyonunu uygulayınız. log dönüşümünün verilerinizi nasıl daha az çarpık hâle getirdiğinin bir örneği Şekil 5.3'te gösterilmiştir. Bu teknik çoğu durumda performans kazancı sağlasa da her durumda işe yaramaz ve orijinal veriler yerine log-dönüşümlü veriler üzerinde yapılan analizlere karşı dikkatli olmalısınız.[5]

Ölçekleme konusunda dikkat edilmesi gereken iki önemli nokta vardır. Birincisi, yaygın bir veri sızıntısı kaynağı olmasıdır (bu durum 145. sayfadaki "Veri Sızıntısı" kısmında daha ayrıntılı olarak ele alınacaktır). Bir diğer nokta ise genellikle global istatistikler gerektirmesidir. Minimumunu, maksimumunu veya ortalamasını hesaplamak için eğitim verilerinin tamamına veya bir alt kümesine bakmanız gerekir. Çıkarsama sırasında, yeni verileri ölçeklemek için

[5] Changyong Feng, Hongyue Wang, Naiji Lu, Tian Chen, Hua He, Ying Lu ve Xin M. Tu, "LogTransformation and Its Implications for Data Analysis," Shanghai Archives of Psychiatry 26, no. 2 (Nisan 2014): 105–9, *https://oreil.ly/hHJjt*.

eğitim sırasında elde ettiğiniz istatistikleri yeniden kullanırsınız. Yeni veriler eğitim verilerine göre önemli ölçüde değiştiyse bu istatistikler pek kullanışlı olmayacaktır. Bu nedenle, bu değişiklikleri hesaba katmak için modelinizi sık sık yeniden eğitmeniz önemlidir.



Şekil 5.3: *Çoğu durumda, log dönüşümü verilerinizin çarpıklığını azaltmaya yardımcı olabilir.*

5.2.3 Kesikleştirme

Bu teknik tamamlayıcı olması açısından kitaba dahil edilmiştir ancak, uygulamada kesikleştirmenin nadiren yardımcı olduğunu gördüm. Tablo 5.2'deki verilerle birlikte bir model geliştirdiğimizi hayal ediniz. Eğitim esnasında modelimiz "150.000", "50.000", "100.000" gibi yıllık gelir değerlerini görmüştür. Çıkarsama esnasında ise modelimiz yıllık geliri "9.000,50" olan bir veri örneğiyle karşılaşır.

Sezgisel olarak yılda 9.000,50 doların yılda 10.000 dolardan çok farklı olmadığını biliyoruz ve modelimizin her ikisine de aynı şekilde davranmasını istiyoruz. Ama model bunu bilmez. Modelimiz yalnızca 9.000,50'nin 10.000'den farklı olduğunu biliyor ve onlara farklı şekilde muamele edecektir.

Kesikleştirme sürekli bir özniteliği kesikli bir özniteliğe dönüştürme işlemidir. Bu işlem nicemleme (ÇN: Quantization) veya gruplara ayırma (ÇN: Binning) olarak da bilinir. Bu, verilen değerler için kovalar (ÇN: Buckets) oluşturarak yapılır. Yıllık gelir söz konusu olduğunda, bu değerleri aşağıdaki gibi üç kova hâlinde gruplayabilirsiniz:

- Düşük gelir: Yılda 35.000 dolardan az
- Orta gelir Yılda 35.000 ila 100.000 dolar arası
- Üst gelir: Yılda 100.000 dolardan fazla

Sonsuz sayıda olası geliri öğrenmek yerine, modelimiz öğrenmesi çok daha kolay bir görev olan yalnızca üç kategoriyi öğrenmeye odaklanabilir. Bu tekniğin sınırlı eğitim verileriyle daha faydalı olması beklenir.

Tanımı gereği kesikleştirme sürekli öznitelikler için olsa da kesikli öznitelikler için de kullanılabilir. Yaş değişkeni kesiklidir fakat aşağıdaki gibi değerleri kovalar hâlinde gruplamak yine de kullanışlı olabilir:

- 18'den küçük
- 18 ila 22 arasında
- 22 ila 30 arasında
- 30 ila 40 arasında
- 40 ila 65 arasında
- 65 üzeri

Bu yöntemin dezavantajı ise, bu kategorileştirmenin kategori sınırında süreksizlikler oluşturmasıdır. Artık 34.999 dolara 35.000 dolardan tamamen farklı bir biçimde muamele edilmektedir ve 35.000 dolar 100.000 dolarla aynı muameleyi görmektedir. Kategorilerin sınırlarını seçmek o kadar da kolay olmayabilir. Değerlerin histogramlarını çizmeyi deneyebilir ve anlamlı olan sınırları seçebilirsiniz. Genel olarak, ortak fikir, basit nicelikler ve bazen konu uzmanlığı yardımcı olabilir.

5.2.4 Kategorik Öznitelikleri Kodlamak

Sürekli özniteliklerin nasıl kategorik özniteliklere dönüştürüleceğinden bahsettik. Bu kısımda kategorik öznitelikleri nasıl en iyi şekilde ele alacağımızdan bahsedeceğiz.

Üretimdeki verilerle çalışmamış kişiler, kategorilerin statik olduğunu varsayma eğilimindedir, bu da kategorilerin zaman içinde değişmediği anlamına gelir. Bu, birçok kategori için geçerlidir. Örneğin, yaş ve gelir gruplarının değişmesi olası değildir ve tam olarak kaç kategori olduğunu önceden bilirsiniz. Bu kategorileri ele almak kolaydır. Her bir kategoriye bir sayı verirsiniz ve işiniz tamamdır.

Ancak üretimde kategoriler değişir. Kullanıcıların Amazon'dan hangi ürünleri satın almak isteyebileceklerini tahmin etmek için bir öneri sistemi oluşturduğunuzu hayal ediniz. Kullanmak istediğiniz özniteliklerden biri ürün markasıdır.

Tablo 6.1: Üç modelin çıktılarına göre topluluğun çıktısı

Üç modelin çıktıları	Olasılık	Topluluğun çıktısı
Her üçü de doğru	$0,7 \times 0,7 \times 0,7 = 0,343$	Doğru
Sadece ikisi doğru	$(0,7 \times 0,7 \times 0,3) \times 3 = 0,441$	Doğru
Sadece biri doğru	$(0,3 \times 0,3 \times 0,7) \times 3 = 0,189$	Yanlış
Hiçbiri doğru değil	$0,3 \times 0,3 \times 0,3 = 0,027$	Yanlış

bir Transformer modeli, bir yinelemeli sinir ağı ve bir gradyan-hızlandırılmış ağaçtan oluşan topluluk oluşturabilirsiniz.

Topluluk oluşturmanın üç yolu vardır: Torbalama, hızlandırma ve yığma. Performans artışına yardımcı olmaya ek olarak, çeşitli araştırma makalelerine göre hızlandırma ve torbalama gibi topluluk yöntemlerinin, yeniden örneklemeyle birlikte dengesiz veri kümelerine yardımcı olduğu gösterilmiştir.[3] Torbalamadan başlayarak, her üç yöntemi de inceleyeceğiz.

Torbalama

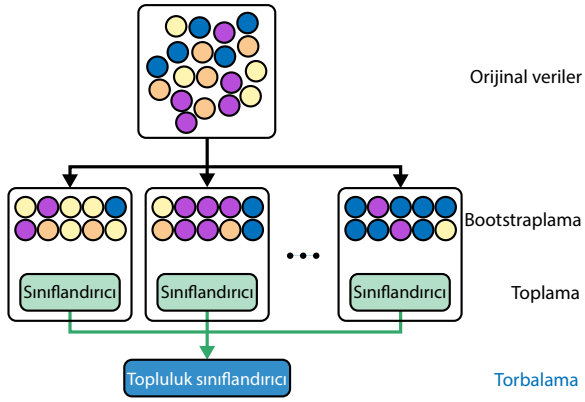
Bootstrap aggregating ifadesinin kısaltması olan torbalama, MÖ algoritmalarının hem eğitim stabilitesini hem de doğruluğunu iyileştirmek için tasarlanmıştır.[4] Torbalama varyansı azaltır ve aşırı uydurmayı engellemeye yardımcı olur.

Bir veri kümesi verildiğinde, tüm veri kümesi üzerinde bir sınıflandırıcı eğitmek yerine, "bootstrap" adı verilen farklı veri kümeleri oluşturmak için yerine koyarak örnekleme (ÇN: Sampling with replacement) gerçekleştirirsiniz ve bu bootstrap kümelerinin her biri üzerinde bir sınıflandırma veya regresyon modeli eğitirsiniz. Yerine koyarak örnekleme, her bootstrap kümesinin diğer bootstrap kümelerinden bağımsız olarak oluşturulmasını sağlar. Şekil 6.3 torbalamanın bir tasviridir.

Problemin sınıflandırma problemi olması hâlinde, nihai tahmine tüm modellerin çoğunluk oyu ile karar verilir. Örneğin, 10 sınıflandırıcı SPAM, 6 model SPAM

[3] Mikel Galar, Alberto Fernandez, Edurne Barrenechea, Humberto Bustince ve Francisco Herrera, "A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches," IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42, no. 4 (Temmuz 2012): 463–84, *https://oreil.ly/ZBlgE*; G. Rekha, Amit Kumar Tyagi ve V. Krishna Reddy, "Solving Class Imbalance Prob- lem Using Bagging, Boosting Techniques, With and Without Using Noise Filtering Method," International Journal of Hybrid Intelligent Systems 15, no. 2 (Ocak 2019): 67–76, *https://oreil.ly/hchzU*.

[4] Burada eğitim stabilitesi, eğitim kaybında daha az dalgalanma anlamına gelmektedir.



Şekil 6.3: *Torbalamanın bir tasviri. Kaynak: Sirakorn'un bir görselinden (https://oreil.ly/KEAPl) uyarlanmıştır.*

DEĞİL şeklinde oy kullanırsa, son tahmin SPAM olur.

Problemin regresyon problemi olması hâlinde, son tahmin bütün modellerin tahminlerinin ortalamasıdır.

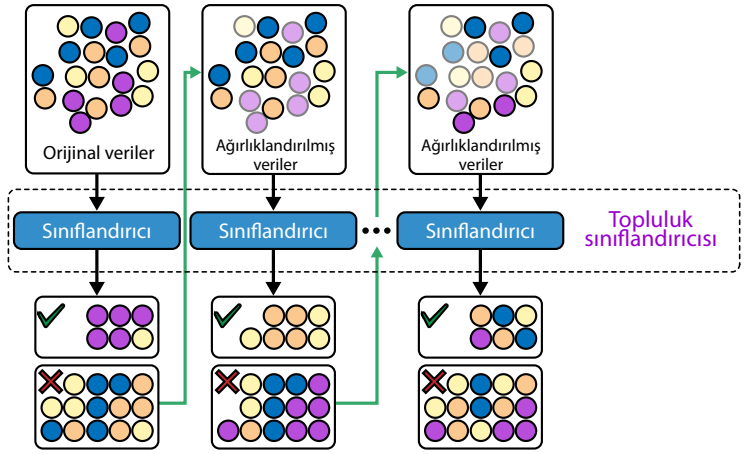
Torbalama genel olarak sinir ağları, sınıflandırma ve regresyon ağaçları ve doğrusal regresyonda alt küme seçimi gibi kararsız yöntemleri iyileştirir. Bununla birlikte, k-en yakın komşular gibi kararlı yöntemlerin performansını biraz düşürebilir.[5]

Bir rastgele orman bir torbalama örneğidir. Rastgele orman hem torbalama hem öznitelik rastgeleliği tarafından oluşturulan bir karar ağaçları koleksiyonudur. Burada her ağaç, kullanmak için özniteliklerin rastgele bir alt kümesinden seçim yapabilir.

Hızlandırma

Hızlandırma (ÇN: Boosting), zayıf öğrenicileri güçlü öğrenicilere dönüştüren yinelemeli bir topluluk algoritmaları ailesidir. Bu topluluktaki her bir öğrenici veri örneklerinin aynı kümesi üzerinde eğitilir ancak bu veri örnekleri iterasyonlar arasında farklı şekilde ağırlıklandırılır. Sonuç olarak, sonraki zayıf öğreniciler, önceki zayıf öğrenicilerin yanlış sınıflandırdığı veri örnekleri üzerine daha çok odaklanır. Şekil 6.4, hızlandırmanın bir tasvirini göstermektedir. Bu tasvir, aşağıdaki adımları içermektedir.

[5] Leo Breiman, "Bagging Predictors," Machine Learning 24 (1996): 123–40, *https://oreil.ly/adzJu*.



Şekil 6.4: *Hızlandırma tasviri. Kaynak: Sirakorn'un bir görselinden (https://oreil.ly/h5cuS) uyarlanmıştır.*

1. İlk zayıf sınıflandırıcıyı orijinal veri kümesi üzerinde eğiterek işe başlarsınız.

2. Birinci sınıflandırıcının ne kadar iyi sınıflandırdığına bağlı olarak veri örnekleri yeniden ağırlıklandırılır. Örneğin, yanlış sınıflandırılan veri örneklerine daha yüksek ağırlık verilir.

3. İkinci sınıflandırıcıyı bu yeniden ağırlıklandırılmış veri kümesi üzerinde eğitiniz. Topluluğunuz artık birinci ve ikinci sınıflandırıcılardan oluşmaktadır.

4. Veri örnekleri, oluşturulan topluluğun onları ne kadar iyi sınıflandırdığına bağlı olarak ağırlıklandırılır.

5. Üçüncü sınıflandırıcıyı bu yeniden ağırlıklandırılmış veri kümesi üzerinde eğitiniz. Üçüncü sınıflandırıcıyı topluluğa ekleyiniz.

6. İhtiyacınız olduğu kadar bu adımları tekrarlayınız.

7. Nihai güçlü sınıflandırıcıyı, mevcut sınıflandırıcıların ağırlıklı bir kombinasyonu olarak oluşturunuz. Daha küçük eğitim hatalarına sahip sınıflandırıcılar daha yüksek ağırlıklara sahiptir.

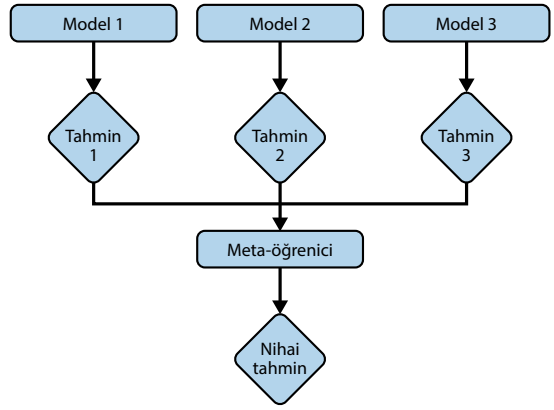
Hızlandırma algoritmasının bir örneği, tipik olarak zayıf karar ağaçlarından bir tahmin modeli üreten gradyan hızlandırma makinesidir (GBM) (ÇN: Gradient Boosting Machine). Bu gradyan hızlandırma makinesi, diğer hızlandırma yöntemlerinin yaptığı gibi aşamalı bir şekilde modeli inşa eder. GBM, keyfi

bir türevlenebilir kayıp fonksiyonunun optimizasyonuna izin vererek diğer hızlandırma yöntemlerini genelleştirir.

GBM'nin bir varyantı olan XGBoost, MÖ yarışmalarını kazanan birçok ekibin tercih ettiği algoritmaydı.[6] Sınıflandırmadan, sıralayamaya, Higgs Boson'unun keşfine kadar çok çeşitli görevlerde kullanılmıştır.[7] Bununla birlikte, birçok ekip, genellikle büyük veri kümelerinde daha hızlı eğitime olanak sağlayan paralel öğrenmeye izin veren dağıtılmış bir gradyan hızlandırma yazılım çerçevesi olan LightGBM'yi (*https://oreil.ly/1qyWf*) tercih etmektedir.

Yığma

Yığma (ÇN: Stacking), temel öğrenicileri eğitim verilerinden eğitmeniz, ardından Şekil 6.5'te gösterildiği gibi nihai tahminlerin çıktısını almak için temel öğrenicilerin çıktılarını birleştiren bir meta-öğrenici oluşturmanız anlamına gelir. Meta-öğrenici bir buluşsal yöntem kadar basit olabilir: Tüm temel öğrenicilerden çoğunluk oyu (sınıflandırma görevlerinde) veya ortalama oyu (regresyon görevlerinde) alırsınız. Lojistik regresyon modeli veya lineer regresyon modeli gibi başka bir model olabilir.



Şekil 6.5: *Üç temel öğreniciden oluşan yığılmış bir topluluğun görselleştirilmesi*

Bir topluluğun nasıl oluşturulacağına dair daha fazla harika tavsiye için Kaggle'ın efsanevi ekiplerinden biri olan MWave'in harika topluluk kılavuzuna (*https://oreil.ly/Nu6G6*) bakınız.

[6] "Machine Learning Challenge Winning Solutions," *https://oreil.ly/YjS8d*.

[7] Tianqi Chen ve Tong He, "Higgs Boson Discovery with Boosted Trees," Proceedings of Machine Learning Research 42 (2015): 69–80, *https://oreil.ly/ysBYO*.

Bir rastgele tohum ayarlayınız

Modelinizin rastgeleliğine katkıda bulunan birçok faktör mevcuttur: Ağırlıklara ilk değer atama, seyreltme, veri karıştırma vb. Rastgelelik, farklı denemeler arasında sonuçları karşılaştırmayı zorlaştırır —performanstaki değişikliğin modeldeki bir değişiklikten ötürü mü yoksa farklı bir rastgele tohumdan ötürü mü olduğu hakkında hiçbir fikriniz yoktur. Bir rastgele tohum ayarlamak, farklı çalıştırmalar arasında tutarlılık sağlar. Ayrıca, hataları yeniden oluşturmanıza ve diğer kişilerin sonuçlarınızı yeniden elde etmesine olanak tanır.

6.1.4 Dağıtılmış Eğitim

Modeller büyüdükçe ve kaynak bakımından yoğun hâle geldikçe şirketler, ölçekte eğitime çok daha fazla önem vermektedir.[11] Büyük bilgi-işlem kaynaklarına düzenli erişim gerektirdiğinden, ölçeklenebilirlik konusunda uzmanlık kazanmak zordur. Ölçeklenebilirlik, bir dizi kitabı hak eden bir konudur. Bu bölüm, ölçekte makine öğrenmesi yapmanın zorluklarını vurgulamak ve projeniz için kaynakları uygun bir biçimde planlamanıza yardımcı olacak bir yapı iskelesi sağlamak için bazı dikkate değer sorunları kapsamaktadır.

Belleğe sığmayan verileri kullanarak bir model eğitmek yaygın bir durumdur. BT taramaları veya genom dizileri gibi tıbbi verilerle uğraşırken bu durum özellikle yaygındır. Büyük dil modellerini eğiten ekipler için çalışıyorsanız (OpenAI, Google, NVIDIA, Cohere) metin verilerinde de bu durum gerçekleşebilir.

Verileriniz belleğe sığmadığında, verileri ön işleme (örneğin sıfır-merkezleme, normalleştirme, beyazlatma (whitening)), verileri karıştırma ve verileri yığınlama için algoritmalarınızın çekirdeğin dışında (ÇN: Out-of-core) ve paralel olarak çalışması gerekir.[12] Verilerinizin bir örneği büyük olduğunda, örneğin, bir makine tek seferde birkaç veri örneği ele alabiliyorsa yalnızca küçük bir yığın büyüklüğüyle çalışmak durumunda kalabilirsiniz ve bu da gradyan inişi tabanlı optimizasyon için kararsızlığa yol açar.

Bazı durumlarda, bir veri örneği o kadar büyüktür ki belleğe bile sığamaz ve sisteminizin daha az bellek ile daha fazla bilgi-işlem yapmasını sağlamak

[11] Çok sayıda kullanıcıya servis veren ürünler için bir modeli servis etmedeki ölçeklenebilirliği de dikkate almanız gerekir. Bu ise bir MÖ projesinin kapsamı dışındadır ve dolayısıyla bu kitapta ele alınmamıştır.

[12] Wikipedia'ya göre, "Çekirdek dışı algoritmalar, tek seferde bir bilgisayarın ana belleğine sığamayacak kadar büyük verileri işlemek için tasarlanmış algoritmalardır.

için bellek ayak izi ve bilgi-işlem dengesini kullanan bir teknik olan gradyan kontrol noktası koyma gibi bir şey kullanmanız gerekir. Açık kaynaklı gradient-checkpointing paketinin sahiplerine göre: "İleri beslemeli modeller için hesaplama süresinde yalnızca %20'lik bir artışla GPU'muza 10 kattan daha büyük modeller sığdırabildik."[13] Bir veri örneği belleğe sığsa bile kontrol noktası koyma tekniğini kullanmak, bir yığına daha fazla veri örneği sığdırmanıza ve bu da modelinizi daha hızlı eğitmenize olanak sağlayabilir.

Veri paralelliği

Makine öğrenmesi modellerini birden çok makinede eğitmek artık bir norm hâline geldi. Modern MÖ yazılım çerçeveleri tarafından desteklenen en yaygın paralelleştirme yöntemi veri paralelliğidir: Verileriniz birden çok makineye bölünür, bu makinelerin hepsinde modeliniz eğitilir ve gradyanlar biriktirilir. Bu, birkaç sorun ortaya çıkarır.

Zorlu bir problem farklı makinelerden gradyanların doğru ve etkili bir şekilde nasıl biriktirileceğidir. Her bir makine kendi gradyanını ürettiğinden eğer modeliniz makinelerin hepsinin çalışmalarını bitirmesini beklerse (SGD-stochastic gradient descent-senkronize stokastik gradyan inişi), geride kalan makineler bütün sistemin yavaşlamasına neden olur ve bu da zaman ve kaynak israfı demektir.[14] Daha fazla işçi demek, belirli bir iterasyonda en az bir işçinin alışılmadık derecede yavaş çalışmasının daha olası olacağı anlamına geldiğinden, geride kalma problemi (ÇN: Straggler problem) makine sayısıyla birlikte büyür. Ancak bu problemi etkin bir biçimde ele alan birçok algoritma bulunmaktadır.[15]

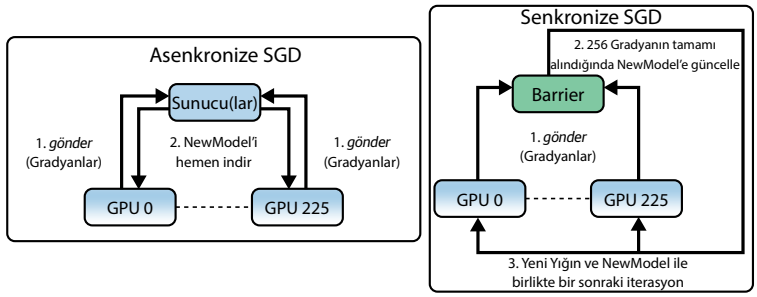
Modeliniz ağırlığı her makineden ayrı olarak gelen gradyanı kullanarak güncelliyorsa (asenkronize SGD), gradyan bayatlaması (ÇN: Gradient staleness) bir sorun hâline gelebilir. Çünkü bir makineden gelen gradyanlar, başka bir

[13]Tim Salimans, Yaroslav Bulatov ve katkıda bulunanlar, gradient-checkpointing deposu, 2017, *https://oreil.ly/GTUgC*.

[14]Dipankar Das, Sasikanth Avancha, Dheevatsa Mudigere, Karthikeyan Vaidynathan, Srinivas Sridharan, Dhiraj Kalamkar, Bharat Kaul ve Pradeep Dubey, "Distributed Deep Learning Using Synchronous Stochastic Gradient Descent," arXiv, 22 Şubat 2016, *https://oreil.ly/ma8Y6*.

[15]Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio ve Rafal Jozefowicz, "Revisiting Distributed Synchronous SGD," ICLR 2017, *https://oreil.ly/dzVZ5*; Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz ve Ion Stoica, "Improving MapReduce Performance in Heterogeneous Environments," 8th USENIX Symposium on Operating Systems Design and Implementation, *https://oreil.ly/FWswd*; Aaron Har- lap, Henggang Cui, Wei Dai, Jinliang Wei, Gregory R. Ganger, Phillip B. Gibbons, Garth A. Gibson ve Eric P. Xing, "Addressing the Straggler Problem for Iterative Convergent Parallel ML" (SoCC '16, Santa Clara, CA, 5–7 Ekim 2016), *https://oreil.ly/wZgOO*.

makineden gradyanlar gelmeden önce ağırlıkların değişmesine neden olmuştur.[16] Senkronize SGD ve asenkronize SGD arasındaki fark Şekil 6.6'da açıklanmıştır.



Şekil 6.6: *Veri paralelliği için asenkronize SGD'ye karşı senkronize SGD. Kaynak: Jim Dowling'in bir görselinden uyarlanmıştır.*[17]

Teoride, asenkronize SGD yakınsar fakat senkronize SGD'den daha fazla adım gerektirir. Bununla birlikte, uygulamada, güncellenecek ağırlık sayısı fazla olduğunda, gradyan güncellemeleri seyrek olma eğilimindedir. Yani çoğu gradyan güncellemesi, parametrelerin yalnızca küçük bir kısmını modifiye eder ve farklı makinelerden gelen iki gradyan güncellemesinin aynı ağırlıkları modifiye etmesi daha az olasıdır. Gradyan güncellemeleri seyrek olduğunda, gradyan bayatlaması daha az problem olur ve model hem senkronize hem de asenkronize SGD için benzer şekilde yakınsar.[18]

Başka bir problem ise modelinizi birden çok makineye yaymanın, yığın büyüklüğünüzün çok büyük olmasına neden olabilmesidir. Bir makine 1.000 büyüklüğünde bir yığın işlerse, 1.000 makine 1 milyon büyüklüğünde bir yığın işler (OpenAI'nin GPT-3 175B'si 2020'de 3,2 milyon büyüklüğünde bir yığın kullanmıştır).[19] Hesaplamayı basitleştirmek gerekirse bir makine üzerinde bir epoku eğitmek 1 milyon adım alırsa, 1.000 makine üzerinde eğitmek sadece 1.000 adım alabilir. Daha sezgisel bir yaklaşım ise her bir adımda daha fazla öğrenmeye sebep olması için öğrenme oranını artırmaktır. Fakat kararsız yakınsamaya yol açacağından, öğrenme oranını çok büyük de yapamayız. Uygu-

[16] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato vd., "Large Scale Distributed Deep Networks," NIPS 2012, *https://oreil.ly/EWPun*.

[17] Jim Dowling, "Distributed TensorFlow," O'Reilly Media, 19 Aralık 2017, *https://oreil.ly/VYlOP*.

[18] Feng Niu, Benjamin Recht, Christopher Ré ve Stephen J. Wright, "Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent," 2011, *https://oreil.ly/sAEbv*.

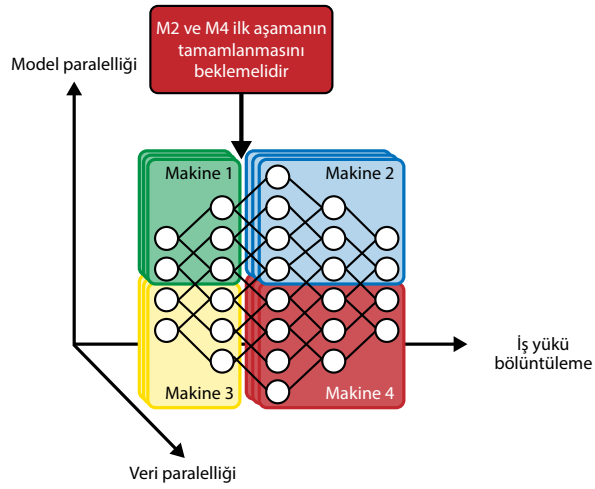
[19] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan vd., "Language Models Are Few-Shot Learners," arXiv, 28 Mayıs 2020, *https://oreil.ly/qjg2S*.

lamada, yığın büyüklüğünü belirli bir noktayı aşacak şekilde arttırmak, azalan getiriler sağlar.[20]

Sonuncusu fakat son derece önemli bir başka problem ise aynı model kurulumuyla ana işçinin bazen diğer işçilerden daha fazla kaynak kullanmasıdır. Durum buysa, tüm makinelerden en iyi şekilde yararlanmak için aralarındaki iş yükünü dengelemenin bir yolunu bulmanız gerekir. En etkili yol olmasa da en kolay yol, ana işçi için daha küçük bir yığın büyüklüğü ve diğer işçiler için daha büyük bir yığın büyüklüğü kullanmaktır.

Model paralelliği

Veri paralelliğiyle her bir işçi tüm modelin bir kopyasına sahiptir ve bu işçi, modelin bu kopyası için gerekli olan bütün hesaplamaları yapar. Model paralelliği ise Şekil 6.7'de gösterildiği gibi modelinizin farklı bileşenleri farklı makinelerde eğitildiğinde söz konusu olur. Örneğin, makine 0 ilk iki katmanın hesaplamasını gerçekleştirirken, makine 1 sonraki iki katmanı ele alır veya bazı makineler ileriye geçişi (ÇN: Forward pass) halledebilirken diğerleri geriye geçişi (ÇN: Backward pass) halledebilir.



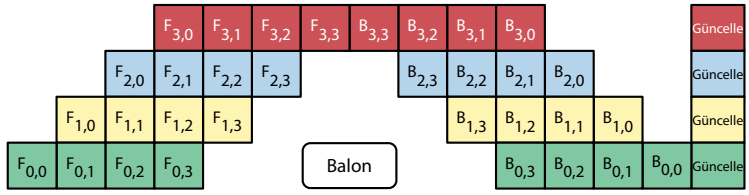
Şekil 6.7: *Veri paralelliği ve model paralelliği. Kaynak: Jure Leskovec'in bir görselinden uyarlanmıştır.*[21]

[20]Sam McCandlish, Jared Kaplan, Dario Amodei ve OpenAI Dota Team, "An Empirical Model of LargeBatch Training," arXiv, 14 Aralık 2018, *https://oreil.ly/mcjbV*; Christopher J. Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig ve George E. Dahl, "Measuring the Effects of Data Parallelism on Neural Network Training," Journal of Machine Learning Research 20 (2019): 1–49, *https://oreil.ly/YAEOM*.

Model paralelliği yanıltıcı olabilir çünkü bazı durumlarda paralellik, modelin farklı parçalarının farklı makinelerde paralel olarak yürütülmesi anlamına gelmez. Örneğin, modeliniz büyük bir matris ise ve bu matris iki makinede ikiye bölünmüşse bu durumda bu iki yarım paralel olarak yürütülebilir. Bununla birlikte, modeliniz bir sinir ağıysa ve ilk katmanı makine 1'e ve ikinci katmanı makine 2'ye koyarsanız ve katman 2'nin yürütülmesi için katman 1'den çıktılara ihtiyacı varsa, makine 2'nin çalışması için önce makine 1'in bitmesini beklemesi gerekir.

İletim hattı paralelliği, bir modelin farklı makineler üzerindeki farklı bileşenlerinin daha fazla paralel olarak çalışmasını sağlamaya yönelik zekice bir tekniktir. Bu paralelliğin birden çok varyantı mevcuttur ancak ana fikir, her makinenin hesaplamasını birden çok parçaya ayırmaktır. Makine 1 hesaplamasının ilk parçasını bitirdiğinde, sonucu makine 2'ye gönderir ve ardından ikinci parçaya geçer ve bu böyle devam eder. Makine 2 artık ilk parça üzerindeki hesaplamasını yürütürken, makine 1 hesaplamasını ikinci parçada yürütür.

Bunu somutlaştırmak için dört farklı makinenizin olduğunu ve birinci, ikinci, üçüncü ve dördüncü katmanların sırasıyla makine 1, 2, 3 ve 4 üzerinde olduğunu düşününüz. İletim hattı paralelliği ile birlikte her bir mini-yığın dört mikro-yığına ayrılır. Makine 1 ilk mikro-yığın üzerinde ilk katmanı hesaplar ve ardından makine 1 ikinci mikro-yığın üzerinde ilk katmanı hesaplarken, makine 2, makine 1'in sonuçları üzerinde ikinci katmanı hesaplar ve bu böyle devam eder. Şekil 6-8, dört makine üzerinde paralelliğin neye benzediğini göstermektedir. Her bir makine, bir sinir ağının bir bileşeni için ileriye geçişi ve geriye geçişi çalıştırır.



Şekil 6.8: *Dört makine üzerindeki bir sinir ağı için iletim hattı paralelliği. Her bir makine bir sinir ağının bir bileşeni için hem ileriye geçişi (F) (ÇN: Forward) hem geriye geçişi (B) (ÇN: Backward) çalıştırır. Kaynak: Huang ve arkadaşlarının bir görselinden uyarlanmıştır.*[22]

Model paralelliği ve veri paralelliği birbirini dışlamaz. Birçok şirket, donanımlarını daha iyi kullanmak için her iki yöntemi de kullanır. Ancak her iki

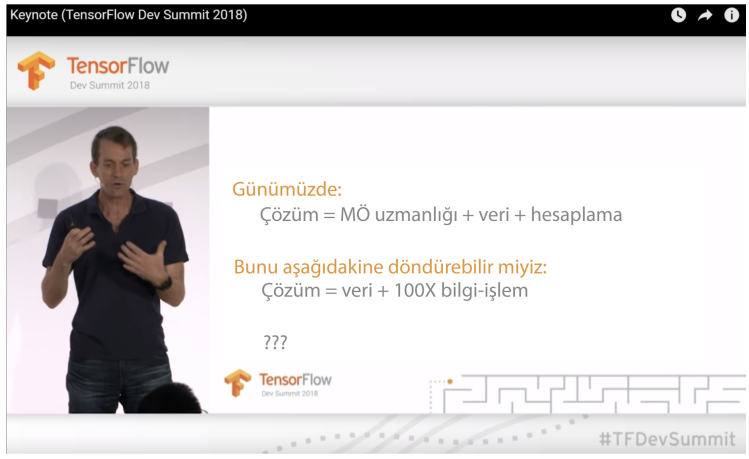
[21] Jure Leskovec, Mining Massive Datasets course, Stanford, lecture 13, 2020, *https://oreil.ly/gZcja*.

[22] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee vd., "GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism," arXiv, 25 Temmuz 2019, *https://oreil.ly/wehkx*.

yöntemi kullanmak için gerçekleştirilecek bir kurulum önemli mühendislik çabası gerektirebilir.

6.1.5 AutoML

İyi bir makine öğrenmesi araştırmacısının, kendi kendini tasarlayacak kadar akıllı bir yapay zekâ algoritması tasarlayarak işini otomatikleştirecek biri olduğuna dair bir şaka var. Bu şaka, Jeff Dean'in sahne aldığı ve Google'ın makine öğrenmesi uzmanlığını 100 kat daha fazla bilgi-işlem gücüyle değiştirmeyi ve AutoML'yi topluluğun heyecan ve dehşetiyle tanıştırmayı amaçladığını açıkladığı TensorFlow Dev Summit 2018'e kadar komikti. 100 makine öğrenmesi araştırmacısı/mühendisinden oluşan bir gruba çeşitli modellerle uğraşmaları ve sonunda optimal olmayan bir model seçmeleri için ödeme yapmak yerine, neden bu parayı en uygun modeli aramak için kullanılabilecek bilgi-işlem gücünde harcamıyorsunuz? Etkinliğin kaydından alınmış bir ekran görüntüsü Şekil 6.9'da gösterilmiştir.



Şekil 6.9: *Jeff Dean, TensorFlow Dev Summit 2018'de Google'ın AutoML'ni açıklıyor.*

Yumuşak AutoML: Hiperparametre ayarlama

AutoML, gerçek dünyadaki problemleri çözmek için makine öğrenmesi algoritmaları bulma sürecini otomatikleştirmeyi ifade eder. Üretimde AutoML'nin kolay ve en popüler biçimi hiperparametre ayarlamadır. Bir hiperparametre, öğrenme sürecini kontrol etmek için kullanılan, kullanıcılar tarafından sağlanan bir parametredir; örneğin, öğrenme oranı, yığın büyüklüğü, gizli katmanların sayısı, gizli birimlerin sayısı, seyreltme olasılığı, Adam optimize edicisinde β_1

Tüm bu kavramlar, kolaylıkla bakımı yapılabilen ve değişen ortamlara adapte olabilen bir MÖ sistemi tasarlamamıza olanak tanır.

Bu bölüm yazarken en çok heyecanlandığım bölümdür ve umarım sizi de heyecanlandırabilirim!

9.1 Aralıksız Öğrenme

"Aralıksız öğrenme" ifadesini duyduğunda birçok kişinin aklına, bir modelin üretimde gelen her örnekle kendini güncellediği eğitim paradigması gelir. Çok az şirket bunu gerçekten yapmaktadır. İlk olarak, modeliniz bir sinir ağı ise gelen her veri örneğiyle öğrenme, onu yıkıcı unutmaya karşı duyarlı hâle getirir. Yıkıcı unutma, bir sinir ağının yeni bilgiler öğrendikten sonra daha önce öğrendiği bilgileri tamamen ve aniden unutma eğilimini ifade eder.[1]

İkinci olarak, aralıksız öğrenme, eğitimi daha pahalı hâle getirebilir; günümüzde çoğu donanım arka ucu yığın işleme için tasarlanmıştır. Bu nedenle tek seferde yalnızca tek bir veri örneğinin işlenmesi, büyük bir bilgi-işlem gücü israfına neden olur ve veri paralelliğinden yararlanamaz.

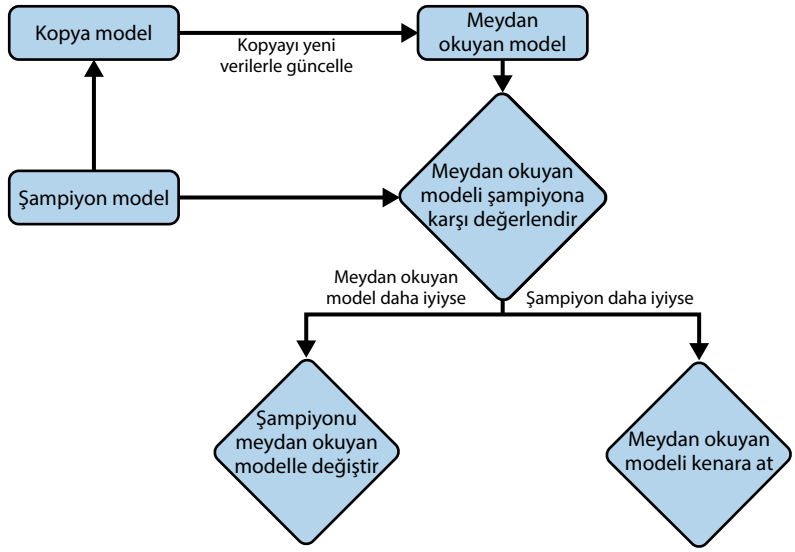
Üretimde aralıksız öğrenmeyi kullanan şirketler, modellerini mikro-yığınlar hâlinde günceller. Örneğin, mevcut modellerini her 512 veya 1.024 veri örneğinden sonra güncelleyebilirler; her bir mikro-yığındaki optimum veri örneği miktarı göreve bağlıdır.

Güncellenen model, değerlendirilene kadar dağıtılmamalıdır. Bu, mevcut modelde doğrudan değişiklik yapmamanız gerektiği anlamına gelir. Bunun yerine, mevcut modelin bir kopyasını oluşturursunuz ve yeni veriler üzerinde bu kopyayı güncellersiniz ve güncellenen kopyanın daha iyi olduğu kanıtlanırsa mevcut modeli bu güncellenmiş kopyayla değiştirirsiniz. Mevcut modele şampiyon model, güncellenmiş kopyaya meydan okuyan model denir. Bu süreç Şekil 9.1'de gösterilmiştir. Bu, anlaşılması için sürecin aşırı basitleştirilmiş hâlidir. Gerçekte ise bir şirketin aynı anda birden fazla meydan okuyan modeli olabilir ve başarısız olmuş bir meydan okuyan modeli ele almak, onu bir kenara atmaktan çok daha sofistikedir.

Yine de, "aralıksız öğrenme" terimi, insanların modelleri çok sık, örneğin her 5 veya 10 dakikada bir güncellediklerini hayal etmelerine neden olur. Pek çok kişi, çoğu şirketin iki nedenden ötürü modellerini bu kadar sık güncellemeye ihtiyaç duymadığını iddia eder. İlk olarak, bu yeniden eğitim zamanlamasının anlamlı olması için yeterli trafiğe (yani, yeterli yeni veriye) sahip değillerdir. İkincisi,

[1] Joan Serrà, Dídac Surís, Marius Miron ve Alexandros Karatzoglou, "Overcoming Catastrophic Forgetting with Hard Attention to the Task," arXiv, 4 Ocak 2018, *https://oreil.ly/P95EZ*.

modelleri o kadar hızlı bozunmaz. Onlara katılıyorum. Yeniden eğitim zamanlamasını bir haftadan bir güne değiştirmek herhangi bir fayda sağlamıyorsa ve daha fazla ek yüke neden oluyorsa, bunu yapmaya gerek yoktur.



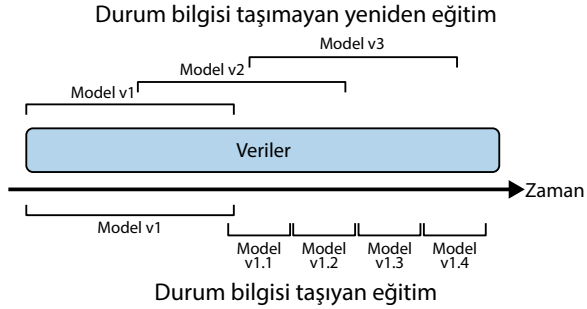
Şekil 9.1: *Aralıksız öğrenmenin üretimde nasıl çalışacağının basitleştirilmiş hâli. Gerçekte ise başarısız olmuş bir meydan okuyan modeli ele almak, onu bir kenara atmaktan çok daha sofistikedir.*

9.1.1 Durum Bilgisi Taşıyan Eğitime Karşı Durum Bilgisi Taşımayan Yeniden Eğitim

Bununla birlikte, aralıksız öğrenme, yeniden eğitim sıklığıyla ilgili değil, modelin yeniden eğitilme şekliyle ilgilidir. Çoğu şirket durum bilgisi taşımayan yeniden eğitim yapar; model her defasında sıfırdan eğitilir. Aralıksız öğrenme, durum bilgisi taşıyan eğitime de izin vermek anlamına gelir; model, yeni veriler üzerinde eğitime devam eder.[2] Durum bilgisi taşıyan eğitim, ince-ayar çekme veya artımlı öğrenme olarak da bilinir. Durum bilgisi taşımayan yeniden eğitim ve durum bilgisi taşıyan eğitim arasındaki fark Şekil 9.2'de görselleştirilmiştir.

Durum bilgisi taşıyan eğitim, modellerinizi daha az veriyle güncelleyebilmenize olanak tanır. Bir modeli sıfırdan eğitmek, aynı modele ince-ayar çekmekten çok daha fazla veri gerektirir. Örneğin, modelinizi sıfırdan yeniden eğitirseniz, son üç aydaki tüm verileri kullanmanız gerekebilir. Ancak dünkü kontrol noktasından modelinize ince ayar çekerseniz yalnızca son güne ait verileri kullanmanız gerekir.

[2] "Durum bilgisi taşıyan yeniden eğitim" yerine "durum bilgisi taşıyan eğitim" denilmiştir çünkü burada yeniden eğitim yoktur. Model en son durumdan itibaren eğitime devam eder.



Şekil 9.2: *Durum bilgisi taşıyan eğitime karşı durum bilgisi taşımayan yeniden eğitim.*

Grubhub, durum bilgisi taşıyan eğitimin, modellerinin daha hızlı yakınsamasını sağladığını ve daha az bilgi-işlem gücü gerektirdiğini bulmuştur. Günlük durum bilgisi taşımayan yeniden eğitimden, günlük durum bilgisi taşıyan eğitime geçmek, eğitim için harcanan bilgi-işlem maliyetini 45 kez azaltmış ve satın alma oranlarını %20 arttırmıştır.[3]

Genellikle gözden kaçan güzel bir özellik, durum bilgisi taşıyan eğitimle veri depolamayı tamamen önlemenin mümkün olabileceğidir. Geleneksel durum bilgisi taşımayan yeniden eğitimde, bir modelin birden çok eğitim iterasyonu sırasında bir veri örneği yeniden kullanılabilir, bu da verilerin saklanması gerektiği anlamına gelir. Bu, özellikle katı gizlilik gereksinimleri olan veriler için her zaman mümkün değildir. Durum bilgisi taşıyan eğitim paradigmasında, Şekil 9.2'de gösterildiği gibi her model güncellemesi yalnızca taze veriler kullanılarak eğitilir, bu nedenle eğitim için bir veri örneği yalnızca bir kez kullanılır. Bu, verileri kalıcı bir depoda depolamak zorunda kalmadan modelinizi eğitmenin mümkün olduğu anlamına gelir; bu da veri gizliliğiyle ilgili birçok endişeyi ortadan kaldırmaya yardımcı olur. Ancak bu göz ardı ediliyor çünkü günümüzün "hadi her şeyi takip edelim" uygulaması hâlâ birçok şirketi verileri çöpe atma konusunda isteksiz hâle getirmektedir.

Durum bilgisi taşıyan eğitim, sıfırdan eğitimin olmayacağı anlamına gelmez. Durum bilgisi taşıyan eğitimi en başarılı şekilde kullanan şirketler, zaman zaman modellerini kalibre etmek için büyük miktarda veri üzerinde sıfırdan eğitim gerçekleştirirler. Alternatif olarak, modellerini durum bilgisi taşıyan eğitimle paralel olarak sıfırdan eğitebilir ve ardından parametre sunucusu gibi teknikler kullanarak her iki güncellenmiş modeli birleştirebilirler.[4]

[3] Alex Egg, "Online Learning for Recommendations at Grubhub," arXiv, 15 Temmuz 2021, *https://oreil.ly/FBBUw*.

[4] Mu Li, Li Zhou, Zichao Yang, Aaron Li, Fei Xia, David G. Andersen ve Alexander Smola, "Parameter Server for Distributed Machine Learning" (NIPS Workshop on Big Learning, Lake Tahoe, CA, 2013), *https://oreil.ly/xMmru*.

Altyapınız hem durum bilgisi taşımayan yeniden eğitime hem de durum bilgisi taşıyan eğitime olanak sağlayacak şekilde kurulduktan sonra eğitim sıklığı sadece bir düğmeyi çevirme meselesidir. Modelinizi saatte bir, günde bir veya bir dağılım kayması tespit edildiğinde güncelleyebilirsiniz. Optimal yeniden eğitim zamanlamasının nasıl bulunacağından 299. sayfadaki "Modellerinizi Hangi Sıklıkla Güncellemelisiniz" kısmında bahsedeceğiz.

Aralıksız öğrenme, bir veri bilimcisi veya MÖ mühendisi olarak sizin, modellerinizi sıfırdan veya ince ayar çekerek gerektiğinde güncellemenize ve bu güncellemeyi hızlı bir şekilde dağıtmanıza olanak sağlayacak şekilde altyapıyı kurmakla ilgilidir.

Şunu merak edebilirsiniz: Durum bilgisi taşıyan eğitim kulağa hoş geliyor ancak modelime yeni bir öznitelik veya başka bir katman eklemek istersem bu nasıl çalışır? Bunu cevaplamak için iki farklı model güncellemesini ayırt etmeliyiz:

Model iterasyonu

> Mevcut model mimarisine yeni bir öznitelik eklenir veya model mimarisi değiştirilir.

Veri iterasyonu

> Model mimarisi ve öznitelikler aynı kalır fakat bu modeli yeni veriler ile tazelersiniz.

Model mimarinizi değiştirmek veya yeni bir öznitelik eklemek, sonuçta ortaya çıkan modelin sıfırdan eğitilmesini gerektirdiğinden, bugün itibariyle durum bilgisi taşıyan eğitim çoğunlukla veri iterasyonu için uygulanmaktadır. Bilgi aktarımı (Google, 2015) (ÇN: Knowledge transfer, *https://oreil.ly/lp0GB*) ve model cerrahisi (OpenAI, 2019) (ÇN: Model surgery, *https://oreil.ly/SU0F1*) gibi teknikler kullanarak model iterasyonu için sıfırdan eğitimi baypas etmenin mümkün olabileceğini gösteren araştırmalar mevcuttur. OpenAI'ye göre "cerrahi işlem, modelin hangi kısımlarının değişmediğini ve hangi kısımlarının yeniden başlatılması gerektiğini tespit etmek için bir seçim sürecinden sonra eğitilmiş ağırlıkları bir ağdan diğerine aktarır."[5] Birkaç büyük araştırma laboratuvarı bununla ilgili denemeler yapmıştır ancak endüstrideki net sonuçlardan haberdar değilim.

[5] Jonathan Raiman, Susan Zhang ve Christy Dennison, "Neural Network Surgery with Sets," arXiv, 13 Aralık 2019, *https://oreil.ly/SU0F1*.

Terminoloji Belirsizliği

"Çevrim içi öğrenme" terimi yerine "aralıksız öğrenme" terimini kullanıyorum çünkü "çevrim içi öğrenme" dediğimde insanlar genellikle çevrim içi (internet üzerinden) verilen eğitimi (ÇN: Online education) düşünüyor. Google'da "çevrim içi öğrenme" diye aratırsanız, en üstte gözüken sonuçlar muhtemelen çevrim içi derslerle (ÇN: Online courses) ilgili olacaktır.

Bazı insanlar, bir modelin gelen her yeni veri örneğinden öğrendiği belirli bir ortama atıfta bulunmak için "çevrim içi öğrenme" ifadesini kullanır. Böyle bir ortamda aralıksız öğrenme, çevrim içi öğrenmenin bir genellemesidir.

Ayrıca, "sürekli öğrenme" (ÇN: Continuous learning) terimi yerine "aralıksız öğrenme" terimini kullanıyorum. Sürekli öğrenme, modelinizin gelen her bir veri örneğiyle sürekli olarak öğrendiği rejimi ifade ederken, aralıksız öğrenmede öğrenme bir dizi yığın veya mikro-yığınlar hâlinde yapılır.

Sürekli öğrenme bazen, aralıksız öğrenme ile yakından ilgili olan makine öğrenmesinin sürekli teslimini (ÇN: Continuous delivery, CD) ifade etmek için kullanılır. Çünkü her ikisi de şirketlerin makine öğrenmesi modellerinin iterasyon döngüsünü hızlandırmasına yardımcı olur. Ancak aradaki fark şu ki bu anlamda kullanıldığında "sürekli öğrenme" DevOps perspektifinden, sürekli teslimat için iletim hattının kurulmasıyla ilgiliyken, "aralıksız öğrenme" makine öğrenmesi perspektifindendir.

"Sürekli öğrenme" teriminin belirsizliğinden dolayı, topluluğun bu terimden tamamen uzak durmasını umuyorum.

9.1.2 Neden Aralıksız Öğrenme?

Aralıksız öğrenmenin, modellerinizi güncelleyebilmeniz ve bu değişiklikleri istediğiniz kadar hızlı dağıtabilmeniz için altyapıyı kurmakla ilgili olduğunu tartıştık. Ancak modellerinizi istediğiniz kadar hızlı güncelleme yeteneğine neden ihtiyacınız olsun ki?

Aralıksız öğrenmenin ilk kullanım durumu, özellikle kaymalar aniden gerçekleştiğinde, veri dağılımı kaymalarıyla mücadele etmektir. Lyft gibi bir yolculuk paylaşımı hizmetinin fiyatlarını belirlemek için bir model oluşturduğunuzu hayal ediniz.[6] Geçmişe baktığımızda, belirli bir mahallede bir Perşembe akşamı yolculuk talebi düşüktür, bu nedenle model, sürücülerin yola koyulmasını daha az cazip hâle getiren düşük yolculuk fiyatları tahmin etmektedir. Ancak bu

[6]Bu tip bir probleme "dinamik fiyatlandırma" da denilir.

10 MLOps için Altyapı ve Araçlar

Bölüm 4'ten Bölüm 6'ya kadar, MÖ sistemleri geliştirmeye yarayan mantıktan bahsettik. Bölüm 7'den Bölüm 9'a kadar ise bir MÖ sistemini dağıtmak, izlemek ve sürekli olarak güncellemekle ilgili hususları tartıştık. Şimdiye kadar, MÖ uygulayıcılarının bu mantığı uygulamak ve bu hususları gerçekleştirmek için ihtiyaç duydukları tüm araçlara ve altyapıya erişebildiklerini varsaydık. Ancak bu varsayım gerçek olmaktan uzaktır. Birçok veri bilimcisi bana MÖ sistemleri için yapmak istedikleri doğru şeyleri bildiklerini ancak altyapılarının bunları yapmayı mümkün kılacak şekilde kurulmadığı için yapamadıklarını söylemiştir.

MÖ sistemleri komplekstir. Bir sistem ne kadar kompleks ise iyi bir altyapıdan o kadar fazla faydalanabilir. Altyapı, doğru kurulduğunda, süreçlerin otomatikleştirilmesine yardımcı olabilir, özelleştirilmiş bilgi ihtiyacını ve mühendislik süresini azaltır. Bu da MÖ uygulamalarının geliştirilmesini ve teslimatını hızlandırabilir, hataların oluşabileceği yüzey alanını küçültebilir ve yeni kullanım senaryolarını mümkün kılabilir. Ancak yanlış kurulduğunda, altyapının kullanımı sancılı ve değiştirilmesi pahalıdır. Bu bölümde, ML sistemleri için altyapının nasıl kurulacağını tartışacağız.

Konuya dalmadan önce, her şirketin altyapı ihtiyaçlarının farklı olduğunu belirtmek önemlidir. Sizin için gerekli olan altyapı, geliştirdiğiniz uygulama sayısına ve uygulamaların ne kadar özelleşmiş olduğuna bağlıdır. Spektrumun bir ucunda, üç aylık planlama toplantısında sunmak üzere gelecek yıl sahip olacakları yeni kullanıcı sayısını tahmin etmek gibi geçici iş analitiği için MÖ kullanan şirketler vardır. Bu şirketlerin muhtemelen herhangi bir altyapıya yatırım yapmaları gerekmeyecektir. Jupyter Notebooks, Python ve Pandas bu şirketlerin en iyi arkadaşları olacaktır. Arkadaşlarınıza göstermek için nesne tespitine yönelik bir Android uygulaması gibi yalnızca basit bir MÖ kullanım senaryonuz varsa muhtemelen herhangi bir altyapıya da ihtiyacınız olmaz; yalnızca TensorFlow Lite gibi Android uyumlu bir MÖ yazılım çerçevesine ihtiyacınız vardır.

Spektrumun diğer ucunda ise benzersiz gereksinimleri olan uygulamalar üzerinde çalışan şirketler bulunur. Örneğin, sürücüsüz arabalar benzersiz doğruluğa ve gecikme gereksinimlerine sahiptir. Algoritma milisaniyeler içerisinde yanıt verebilmelidir ve yanlış bir tahmin ciddi kazalara yol açabileceğinden, algoritmanın doğruluğu mükemmele yakın olmalıdır. Benzer şekilde, çoğu şirket Google'ın yaptığı gibi saatte 234 milyon arama sorgusuna karşılık gelen saniyede 63.000 arama sorgusunu işlemediğinden, Google Arama benzersiz bir ölçek gereksinimine sahiptir.[1] Bu şirketlerin muhtemelen kendi özelleştirilmiş altyapılarını geliştirmeleri gerekecektir. Google, dahili altyapılarının büyük bir bölümünü arama için geliştirmiştir; Tesla ve Waymo gibi sürücüsüz araba şirketleri de öyle.[2] Özelleştirilmiş altyapının bir kısmının daha sonra herkese açılması ve diğer şirketler tarafından benimsenmesi yaygın bir durumdur. Örneğin Google, dahili bulut altyapısını herkese açarak Google Cloud Platform'u (*https://oreil.ly/0g02L*) ortaya çıkarmıştır.

Spektrumun ortasında ise makine öğrenmesini makul ölçekte birden çok yaygın uygulama (bir dolandırıcılık tespiti modeli, bir fiyat optimizasyonu modeli, bir müşteri kaybı tahmini modeli, bir öneri sistemi vb.) için kullanan şirketlerin çoğunluğu yer alır. "Makul ölçek", günlük petabayt yerine gigabayt ve terabayt düzeyindeki verilerle çalışan şirketleri ifade eder. Bu şirketlerin veri bilimi ekipleri 10 ila yüzlerce mühendisten oluşabilir.[3] Bu kategori, 20 kişilik bir startup'tan, FAAAM ölçeğinde olmasa da Zillow ölçeğindeki bir şirkete kadar herhangi bir şirketi içerebilir.[4] Örneğin, 2018'de Uber, veri gölüne günde onlarca terabayt veri ekliyordu ve Zillow'un en büyük veri kümesi günde 2 terabayt sıkıştırılmamış veri getiriyordu.[5] Buna karşın, 2014'te bile Facebook günde 4 petabaytlık veri üretiyordu.[6]

[1] Kunal Shah, "This Is What Makes SEO Important for Every Business," Entrepreneur India, 11 Mayıs 2020, *https://oreil.ly/teQlX*.

[2] Tesla'nın makine öğrenmesi için bilgi-işlem altyapısına göz atmak için YouTube'da Tesla AI Day 2021 kaydını (*https://oreil.ly/etH9C*) izlemenizi şiddetle tavsiye ederim.

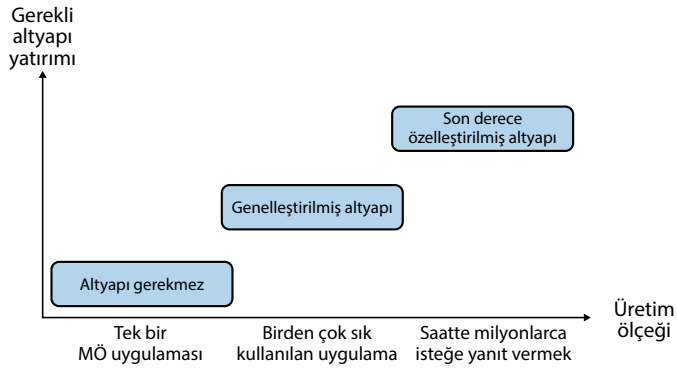
[3] "Makul ölçeğin" tanımı Jacopo Tagliabue'nın "You Do Not Need a Bigger Boat: Recommendations at Reasonable Scale in a (Mostly) Serverless and Open Stack," arXiv, 15 Temmuz 2021, *https://oreil.ly/YNRZQ* isimli makalesinden esinlenilmiştir. Makul ölçek üzerine daha fazla tartışma için Ciro Greco'nun "ML and MLOps at a Reasonable Scale (ÇN: Makul Ölçekte Makine Öğrenmesi ve Makine Öğrenmesi Operasyonları)" (Ekim 2021) (*https://oreil.ly/goPrb*) isimli çalışmasına bakınız.

[4] FAAAM, Facebook, Apple, Amazon, Alphabet, Microsoft'un kısaltmasıdır.

[5] Reza Shiftehfar, "Uber's Big Data Platform: 100+ Petabytes with Minute Latency," Uber Engineering, 17 Ekim 2018, *https://oreil.ly/6Ykd3*; Kaushik Krishnamurthi, "Building a Big Data Pipeline to Process Clickstream Data," Zillow, 6 Nisan 2018, *https://oreil.ly/SGmNe*.

[6] Nathan Bronson ve Janet Wiener, "Facebook's Top Open Data Problems," Meta, 21 Ekim 2014, *https://oreil.ly/p6QjX*.

Spektrumun ortasında yer alan şirketler, giderek daha fazla standartlaştırılan genelleştirilmiş makine öğrenmesi altyapısından büyük olasılıkla faydalanacaktır (bakınız Şekil 10.1). Bu kitapta, makine öğrenmesi uygulamalarının büyük çoğunluğunun altyapısına makul bir ölçekte odaklanacağız.



Şekil 10.1: *Farklı üretim ölçeklerindeki şirketler için altyapı gereksinimleri.*

İhtiyaçlarınız için doğru altyapıyı kurmak üzere, altyapının tam olarak ne anlama geldiğini ve nelerden oluştuğunu anlamak önemlidir. Wikipedia'ya göre fiziksel dünyada "altyapı, evlerin ve firmaların sürdürülebilir işlevselliğini destekleyen temel olanaklar ve sistemler kümesidir."[7] Makine öğrenmesi dünyasında altyapı, makine öğrenmesi sistemlerinin geliştirilmesini ve bakımını destekleyen temel olanaklar kümesidir. Bu bölümde daha önce tartışıldığı gibi "temel olanaklar" olarak nelerin dikkate alınması gerektiği şirketten şirkete büyük ölçüde değişir. Bu bölümde aşağıdaki dört katmanı inceleyeceğiz:

Depolama ve bilgi-işlem

> Depolama katmanı verilerin toplanıp, depolandığı katmandır. Bilgi-işlem katmanı ise bir modeli eğitmek, öznitelikleri hesaplamak, öznitelikleri üretmek vb. gibi MÖ iş yüklerini yerine getirmek için gerekli olan bilgi-işlemi sağlar.

Kaynak yönetimi

> Kaynak yönetimi, mevcut bilgi-işlem kaynaklarınızdan en iyi şekilde yararlanmak için iş yüklerinizi zamanlamak ve orkestra etmek için araçlar içerir. Bu kategorideki araçlara örnek olarak Airflow, Kubeflow ve Metaflow verilebilir.

[7] Wikipedia, s.v. "Infrastructure," *https://oreil.ly/YaIk8*.

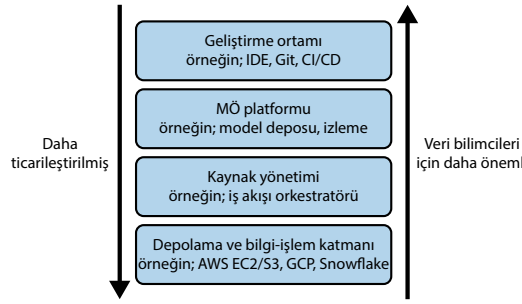
MÖ platformu

> Bu katman, model depoları, öznitelik depoları ve izleme araçları gibi MÖ uygulamalarını geliştirmeye yardımcı olan araçları sağlar. Bu kategorideki araçlara örnek olarak SageMaker ve MLflow verilebilir.

Geliştirme ortamı

> Bu katmana genellikle dev ortamı denilir. Kodun yazıldığı ve denemelerin yürütüldüğü katman bu katmandır. Kodun versiyonlanması ve test edilmesi gerekmektedir. Denemelerin de takip edilmesi gerekmektedir.

Bu dört katman Şekil 10.2'de gösterilmiştir. Veri ve bilgi-işlem, herhangi bir makine öğrenmesi projesi için gereken başlıca kaynaklardır ve bu nedenle depolama ve bilgi-işlem katmanı, makine öğrenme uygulamak isteyen herhangi bir şirket için altyapısal temeli oluşturur. Bu katman aynı zamanda bir veri bilimcisi için en soyut olan katmandır. İlk olarak bu katmandan bahsedeceğiz çünkü bu kaynaklar açıklaması en kolay olanlardır.



Şekil 10.2: *MÖ altyapısının farklı katmanları.*

Dev ortamı, veri bilimcilerinin günlük olarak etkileşime girmesi gereken ortamdır ve bu nedenle onlar için en az soyut olanıdır. İlk olarak bu kategoriyi ele alacağız, ardından veri bilimcileri arasında tartışmalı bir konu olan kaynak yönetimini tartışacağız; insanlar hâlâ bir veri bilimcinin bu katman hakkında bilgi sahibi olması gerekip gerekmediğini tartışmaktadır. "MÖ platformu", farklı bileşenleri hâlâ olgunlaşmakta olan nispeten yeni bir kavram olduğundan, diğer tüm kategorilere aşina olduktan sonra bu kategoriyi en son tartışacağız. Bir makine öğrenmesi platformu, bir şirketin ön yatırım yapmasını gerektirir ancak doğru bir şekilde yapılırsa o şirketteki işle ilgili kullanım durumlarında veri bilimcilerin hayatını çok daha kolay hâle getirebilir.

İki şirket tamamen aynı altyapı ihtiyaçlarına sahip olsa bile sonuçta ortaya çıkan altyapı, şirketlerin altyapıyı kendileri oluşturması veya bu altyapıyı satın almaları kararlarına (yani neyi şirket içerisinde geliştirmek istediklerine karşı neyi dışarıdan satın almak istediklerine) yönelik yaklaşımlarına bağlı olarak farklı görünebilir. Bu bölümün son kısmında, altyapıyı oluşturmaya karşı altyapıyı başkasından satın alma kararlarını ele alacağız ve burada makine öğrenmesi altyapısı için standartlaştırılmış ve birleştirilmiş soyutlamalara yönelik beklentileri de tartışacağız.

Haydi başlayalım!

10.1 Depolama ve Bilgi-işlem

MÖ sistemleri çok miktarda veri ile çalışır ve bu verilerin bir yerlerde depolanması gerekir. Depolama katmanı verilerin toplandığı ve depolandığı katmandır. En basit hâliyle depolama katmanı bir sabit sürücü diski (HDD) veya bir katı hâl diski (SSD) olabilir. Depolama katmanı tek bir yerde –örneğin bütün verileriniz Amazon S3 veya Snowflake'de olabilir– olabildiği gibi birden çok lokasyona da yayılmış olabilir.[8] Depolama katmanınız şirket-içi özel bir veri merkezinde veya bulutta olabilir. Geçmişte şirketler kendi depolama katmanını yönetmeye çalışmış olabilir. Ancak son on yılda, depolama katmanı çoğunlukla ticarileştirilmiş ve buluta taşınmıştır. Veri depolama o kadar ucuz hâle geldi ki çoğu şirket sahip oldukları tüm verileri hiçbir maliyet ödemeden depolamaktadır.[9] Veri katmanını Bölüm 3'te derinlemesine ele aldık, dolayısıyla bu bölümde bilgi-işlem katmanına odaklanacağız.

Bilgi-işlem katmanı, bir şirketin erişebildiği tüm bilgi-işlem kaynaklarını ve bu kaynakların nasıl kullanılabileceğini belirleyen mekanizmayı ifade eder. Kullanılabilir bilgi-işlem kaynaklarının miktarı, iş yüklerinizin ölçeklenebilirliğini belirler. Bilgi-işlem katmanını, işlerinizi yürütecek motor olarak düşünebilirsiniz. En basit hâliyle, bilgi-işlem katmanı tüm hesaplamalarınızı yapan tek bir CPU çekirdeği veya bir GPU çekirdeği olabilir. En yaygın biçimi, AWS Elastic Compute Cloud (EC2) veya GCP gibi bir bulut sağlayıcısı tarafından yönetilen bulutta bilgi-işlemdir.

Bilgi-işlem katmanı genellikle eş zamanlı kullanılmak üzere daha küçük bilgi-işlem birimlerine bölünebilir. Örneğin, bir CPU çekirdeği iki eş zamanlı izleği

[8] Verileri Amazon Redshift ve GCP BigQuery'ye yayılmış olan bir şirket ile karşılaşmıştım ve mühendisleri bu durumdan pek memnun değildi.

[9] Veri sistemlerinden Bölüm 2'de bahsettiğimizden burada sadece veri depolamadan bahsedeceğiz.

Dizin