6. Understanding Machine Learning - Based Malware Detectors :Makine Öğrenimi Tabanlı Kötü Amaçlı Yazılım Algılayıcılarını Anlama

Bu bölümde kendi algılama araçlarınızı yüksek düzeyde geliştirme sürecini öğreneceksiniz.Özellik alanları, karar sınırları, eğitim verileri , yetersiz uyum ve aşırı uyum dahil olmak üzere makine öğreniminnin arkasındaki fikirlere bakalım.Daha sonra dört temel yaklaşıma bakalım:lojistik regrasyon, k en yakın komşular, karar ağaçları ve rastgele orman ve bunların algılama gerçekleştirmek için nasıl uygulanabileceği. Daha sonra, Bölüm 7'de makine öğrenimi sistemlerinin doğruluğunu nasıl değerlendireceğinizi ve Bölüm 8'de Python'da makine öğrenimi sistemlerini nasıl uygulayacağınızı öğrenmek için bu bölümde öğrendiklerinizi kullanacaksınız.

->Steps for Building a Machine Learning–Based Detector: Makine Öğrenimi Tabanlı Bir Dedektör Oluşturma Adımları

Makine öğrenimi ile diğer bilgisayar algoritmaları arasında temel bir fark vardır. Geleneksel algoritmalar bilgisayara ne yapması gerektiğini söylerken, makine öğrenimi sistemleri bir problemi örnekleyerek nasıl çözeceklerini öğrenir. Örneğin, önceden yapılandırılmış bir dizi kuraldan basitçe yararlanmak yerine, makine öğrenimi güvenlik algılama sistemleri, iyi ve kötü dosya örneklerinden öğrenerek bir dosyanın kötü veya iyi olup olmadığını belirlemek için eğitilebilir.

Bir maikne öğrenimi tabanlı algılayıcı oluşturmak için izlediğimiz iş akışı şu şekildedir:

1. Collect: Kötü amaçlı yazılım ve zararlı yazılım örneklerini toplayın. Makine öğrenimi sistemini kötü amaçlı yazılımları tanıması için eğitmek için bu örnekleri (eğitim örnekleri olarak adlandırılır training examples) kullanacağız.
2. Extract: Örneği bir sayı dizisi olarak temsil etmek için her eğitim örneğinden özellikleri çıkarın. Bu adım aynı zamanda makine öğrenimi sisteminizin doğru çıkarımlar yapmasına yardımcı olacak iyi özellikler tasarlamaya yönelik araştırmaları da içerir.
3. Train: Çıkardığımız özellikleri kullanarak kötü amaçlı yazılımları tanıması için makine öğrenimi sistemini eğitin.
4. Test: Algılama sistemimizin ne kadar iyi çalıştığını görmek için yaklaşımı eğitim örneklerimizde bulunmayan bazı veriler üzerinde test edin.

Çeşitli makine öğrenimi algoritmalarının nasıl çalıştığını tartışalım.

->Understanding Feature Spaces and Decision Boundaries: Özellik Uzaylarını ve Karar Sınırlarını Anlama:

İki basit geometrik fikir, tüm makine öğrenimi tabanlı algılama algoritmalarını anlamanıza yardımcı olabilir: geometrik özellik alanı fikri ve bir karar sınırı fikri. Bir özellik alanı, seçtiğiniz özelliklerle tanımlanan geometrik uzaydır ve bir karar sınırı, bu sınırın bir tarafındaki ikili dosyaların kötü amaçlı yazılım olarak tanımlandığı ve diğer tarafındaki ikili dosyaların iyi olarak tanımlandığı geometrik bir yapıdır.  Dosyaları kötü niyetli veya zararsız olarak sınıflandırmak için bir makine öğrenimi algoritması kullandığımızda, örnekleri özellik alanına yerleştirebilmemiz için özellikleri çıkarırız ve ardından dosyaların karar sınırının hangi tarafında olduğunu kontrol ederek dosyaların kötü amaçlı yazılım veya zararlı yazılım olup olmadığını belirleriz.  Kötü amaçlı yazılım ve zararlı yazılım örneklerinden oluşan bir eğitim veri kümemiz olduğunu varsayalım. Şimdi, her ikiliden şu iki özelliği çıkardığımızı varsayalım: sıkıştırılmış görünen dosyanın yüzdesi ve her ikilinin içe aktardığı şüpheli işlevlerin sayısı. Eğitim veri setimizi Şekil 6-1'de gösterildiği gibi görselleştirebiliriz (veriyi arsadaki verileri yapay olarak, örneğin amaçlarla oluşturduğumu unutmayın).

Şekil 6-1: Bu bölümde kullanacağımız, gri noktaların iyi huylu ve siyah noktaların kötü amaçlı yazılım olduğu örnek bir veri kümesinin grafiği

İki özelliğimiz tarafından tanımlanan Şekil 6-1'de gösterilen iki boyutlu uzay, örnek veri setimizin özellik alanıdır. Siyah noktaların (kötü amaçlı yazılımlar) genellikle boşluğun sağ üst kısmında olduğu net bir model görebilirsiniz. Genel olarak, bunlar çoğunlukla grafiğin sol alt kısmında bulunan iyi niyetli yazılıma göre daha şüpheli içe aktarılmış işlev çağrılarına ve daha sıkıştırılmış verilere sahiptir. Bu taslağı görüntüledikten sonra, yalnızca burada kullandığımız iki özelliği temel alan bir kötü amaçlı yazılım tespit sistemi oluşturmanız istendiğini varsayalım. Verilere dayanarak, şu kuralı formüle edebileceğiniz açıktır: Bir ikili hem çok sayıda sıkıştırılmış veriye hem de çok sayıda şüpheli içe aktarılmış işlev çağrısına sahipse, kötü amaçlı yazılımdır ve çok fazla şüpheli içe aktarılmış çağrısı yoksa ne de sıkıştırılmış veri, iyi huylu yazılımdır.

Geometrik terimlerle, bu kuralı, yeterli sıkıştırılmış veriye ve içe aktarılan işlev çağrılarına (kötü amaçlı yazılım olarak tanımlanan) sahip ikili dosyaların satırın üstünde ve ikili dosyaların geri kalanı (iyi huylu yazılım olarak tanımlanır) çizginin altındadır. Şekil 6-2, karar sınırı dediğimiz böyle bir çizgiyi göstermektedir.

Şekil 6-2: Kötü amaçlı yazılımları tespit etmek için bir kural tanımlayan, örnek veri kümemiz aracılığıyla çizilen bir karar sınırı

Çizgiden de görebileceğiniz gibi, siyah (kötü amaçlı yazılım) noktaların çoğu sınırın bir tarafında ve gri (iyi huylu yazılım) örneklerin çoğu karar sınırının diğer tarafındadır. Tüm örnekleri birbirinden ayıran bir çizgi çizmenin imkansız olduğuna dikkat edin, çünkü bu veri kümesindeki siyah ve gri bulutlar birbiriyle örtüşüyor. Ancak bu örneğe baktığımızda, yeni kötü amaçlı yazılım örneklerini ve iyi huylu yazılım örneklerini çoğu durumda doğru şekilde sınıflandıracak bir çizgi çizdik ve bu görüntüdeki verilerde görülen modeli izlediklerini varsayıyoruz.

Şekil 6-2'de, verilerimiz aracılığıyla manuel olarak bir karar sınırı çizdik. Peki ya daha kesin bir karar sınırı istiyorsak ve bunu otomatik bir şekilde yapmak istiyorsak? Makine öğreniminin yaptığı tam olarak budur. Diğer bir deyişle, tüm makine öğrenimi algılama algoritmaları verilere bakar ve ideal karar sınırını belirlemek için otomatik bir süreç kullanır; böylelikle yeni, daha önce görülmemiş veriler üzerinde doğru bir şekilde algılama gerçekleştirme şansı en yüksektir.

Gerçek dünyada yaygın olarak kullanılan bir makine öğrenimi algoritmasının Şekil 6-3'te gösterilen örnek veriler içindeki bir karar sınırını nasıl tanımladığına bakalım. Bu örnek, lojistik regresyon adı verilen bir algoritma kullanır.

Şekil 6-3: Lojistik regresyon modeli eğitilerek otomatik olarak oluşturulan karar sınırı

Gri noktaların zararsız ve siyah noktaların kötü amaçlı yazılım olduğu önceki grafiklerde kullandığımız örnek verileri kullandığımıza dikkat edin. Grafiğin ortasından geçen çizgi, lojistik regresyon algoritmasının verilere bakarak öğrendiği karar sınırıdır. Hattın sağ tarafında, lojistik regresyon algoritması, ikili dosyaların kötü amaçlı yazılım olma olasılığını yüzde 50'den fazla atar ve satırın sol tarafında, bir ikili dosyanın kötü amaçlı yazılım olma olasılığını yüzde 50'den daha az bir olasılık atar. Şimdi arsanın gölgeli bölgelerine dikkat edin. Koyu gri gölgeli bölge, lojistik regresyon modelinin dosyaların kötü amaçlı yazılım olduğundan son derece emin olduğu bölgedir. Lojistik regresyon modelinin gördüğü, bu bölgeye inen özelliklere sahip herhangi bir yeni dosyanın kötü amaçlı yazılım olma olasılığı yüksek olmalıdır. Karar sınırına yaklaştıkça, model ikili dosyaların kötü amaçlı yazılım veya iyi huylu yazılım olup olmadığı konusunda giderek daha az güven duyuyor. Lojistik regresyon, kötü amaçlı yazılımları tespit etme konusunda ne kadar agresif olmak istediğimize bağlı olarak, çizgiyi daha karanlık bölgeye veya daha hafif bölgeye kolayca taşımamıza olanak tanır. Örneğin, onu aşağı taşırsak, daha fazla kötü amaçlı yazılım yakalarız, ancak daha fazla yanlış pozitif alırız. Yukarı taşırsak, daha az kötü amaçlı yazılım yakalarız, ancak daha az yanlış pozitif alırız. Lojistik regresyonun ve diğer tüm makine öğrenme algoritmalarının keyfi olarak yüksek boyutlu özellik alanlarında çalışabileceğini vurgulamak istiyorum. Şekil 6-4, biraz daha yüksek boyutlu bir özellik uzayında lojistik regresyonun nasıl çalıştığını göstermektedir. Bu yüksek boyutlu uzayda, karar sınırı bir çizgi değil, 3B hacimdeki noktaları ayıran bir düzlemdir. Dört veya daha fazla boyuta geçecek olsaydık, lojistik regresyon, kötü amaçlı yazılımı bu yüksek boyutlu uzaydaki iyi huylu yazılım noktalarından ayıran n boyutlu düzlem benzeri bir yapı olan bir hiper düzlem yaratırdı.

Şekil 6-4: Lojistik regresyon ile oluşturulan varsayımsal üç boyutlu bir özellik uzayında düzlemsel bir karar sınırı

Lojistik regresyon nispeten basit bir makine öğrenme algoritması olduğundan, yalnızca çizgiler, düzlemler ve daha yüksek boyutlu düzlemler gibi basit geometrik karar sınırları oluşturabilir. Diğer makine öğrenimi algoritmaları, daha karmaşık olan karar sınırları oluşturabilir. Örneğin, k-en yakın komşu algoritması (kısaca ayrıntılı olarak tartışacağım) tarafından verilen, Şekil 6 5'te gösterilen karar sınırını düşünün.,

Şekil 6-5: k-en yakın komşu algoritması tarafından oluşturulan bir karar sınırı

Gördüğünüz gibi, bu karar sınırı bir düzlem değil: oldukça düzensiz bir yapıdır. Ayrıca, bazı makine öğrenimi algoritmalarının, bu bölgeler bitişik olmasa bile, özellik alanının bazı bölgelerini kötü niyetli ve bazı bölgeleri zararsız olarak tanımlayan ayrık karar sınırları oluşturabileceğini unutmayın. Şekil 6-6, örnek özellik alanımızda daha karmaşık bir kötü amaçlı yazılım ve iyi huylu yazılım modeline sahip farklı bir örnek veri kümesi kullanarak bu düzensiz yapıya sahip bir karar sınırını göstermektedir.

Şekil 6-6: k en yakın komşu algoritması tarafından oluşturulan ayrık bir karar sınırı

Karar sınırı bitişik olmasa da, bu ayrık karar sınırlarını basitçe "karar sınırları" olarak adlandırmak hala yaygın olan makine öğrenimi deyimidir. Farklı türdeki karar sınırlarını ifade etmek için farklı makine öğrenimi algoritmaları kullanabilirsiniz ve ifade gücündeki bu fark, belirli bir proje için bir makine öğrenimi algoritmasını diğerine tercih etmemizin nedenidir.

Artık özellik alanları ve karar sınırları gibi temel makine öğrenimi kavramlarını keşfettiğimize göre, makine öğrenimi uygulayıcılarının bundan sonra neyin gereğinden fazla ve yetersiz uyum olarak adlandırdığını tartışalım.machine learning practitioners call overfitting and underfitting

->What Makes Models Good or Bad: Overfitting and

Underfitting: Modelleri İyi veya Kötü Yapan Nedir: Aşırı Uyum ve Yetersiz Uyum

Makine öğrenimine gereğinden fazla ve yetersiz uyum sağlamanın önemini fazla vurgulayamıyorum. Her iki durumdan da kaçınmak, iyi bir makine öğrenimi algoritmasını tanımlayan şeydir. Makine öğrenimindeki iyi ve doğru algılama modelleri, kuralı kanıtlayan aykırı değerler veya istisnalar dikkatinizi dağıtmadan, kötü amaçlı yazılımları iyi huylu yazılımlardan ayıran şey hakkında eğitim verilerinin söylediği genel eğilimi yakalar.

Underfit modelleri aykırı değerleri görmezden gelir, ancak genel eğilimi yakalayamaz, bu da yeni, daha önce görülmemiş ikili dosyalar üzerinde zayıf doğruluk ile sonuçlanır. Overfit modelleri, genel eğilimi yansıtmayan şekillerde aykırı değerler tarafından dikkati dağılır ve daha önce görülmeyen ikili dosyalar üzerinde zayıf doğruluk sağlar. Makine öğrenimi kötü amaçlı yazılım algılama modelleri oluşturmak, tamamen kötü niyetli kişileri zararsız olanlardan ayıran genel eğilimi yakalamakla ilgilidir.

Bu terimleri açıklamak için Şekil 6-7, 6-8 ve 6-9'daki yetersiz uyan, iyi uyan ve aşırı uyan model örneklerini kullanalım. Şekil 6-7, yetersiz oturan bir modeli göstermektedir.

Şekil 6-7: Yetersiz bir makine öğrenimi modeli

Burada, grafiğin sağ üst bölgesinde siyah nokta (kötü amaçlı yazılım) kümesini ve sol altta gri nokta (iyi huylu yazılım) kümesini görebilirsiniz. Bununla birlikte, makine öğrenimi modelimiz, köşegen eğilimi yakalamadan verileri kabaca ayırarak noktaları ortadan keser. Model genel eğilimi yakalamadığı için uygun olmadığını söylüyoruz.

Ayrıca, modelin çizimin tüm bölgelerinde verdiği yalnızca iki kesinlik tonu olduğuna dikkat edin: gölge koyu gri veya beyazdır. Diğer bir deyişle, model ya özellik alanındaki noktaların kötü niyetli olduğundan ya da zararsız olduklarından kesinlikle emindir. Kesinliğin doğru bir şekilde ifade edilememesi, aynı zamanda bu modelin yetersiz kalmasının bir nedenidir.

Şekil 6-7'deki yetersiz uydurma modelini Şekil 6-8'deki iyi uyan modelle karşılaştıralım.

Şekil 6-8: Uygun bir makine öğrenimi modeli

Bu durumda, model yalnızca verilerdeki genel eğilimi yakalamakla kalmaz, aynı zamanda özellik uzayının hangi bölgelerinin kesinlikle kötü niyetli, kesinlikle zararsız veya gri bir alanda olduğu tahminine göre makul bir kesinlik modeli oluşturur.

Bu grafiğin tepesinden altına doğru uzanan karar çizgisine dikkat edin. Model, kötü amaçlı yazılımı iyi huylu yazılımdan ayıran şeyin ne olduğuna dair basit bir teoriye sahiptir: grafiğin ortasında çapraz bir çentik bulunan dikey bir çizgi. Ayrıca, modelin yalnızca grafiğin sağ üst kısmındaki verilerin kötü amaçlı yazılım olduğundan emin olduğunu ve grafiğin sol alt köşesindeki ikili dosyaların iyi huylu yazılım olduğundan emin olduğunu söyleyen grafikteki gölgeli bölgelere de dikkat edin.

Son olarak, Şekil 6-9'da gösterilen fazla uydurma modelini Şekil 6-7'de gördüğünüz yetersiz uydurma modeliyle ve Şekil 6-8'deki iyi uydurma modeliyle karşılaştıralım.

Şekil 6-9'daki aşırı uyum modeli, verilerdeki genel eğilimi yakalayamamaktadır. Bunun yerine, gri nokta kümesinde (iyi huylu eğitim örnekleri) oluşan bir avuç siyah nokta (kötü amaçlı eğitim örnekleri) dahil olmak üzere verilerdeki istisnaları saplantı haline getirir ve etraflarına karar sınırları çizer. Benzer şekilde, kötü amaçlı yazılım kümesinde meydana gelen bir avuç iyi huylu yazılım örneğine odaklanır ve bunların etrafına da sınırlar çizer.

Bu, onları bu aykırı değerlere yaklaştıran özelliklere sahip olan yeni, daha önce görülmemiş ikili dosyalar gördüğümüzde, makine öğrenimi modelinin neredeyse kesinlikle iyi huylu yazılım olduklarında kötü amaçlı yazılım olduklarını düşüneceği ve bunun tersi anlamına gelir. Pratikte bu, bu modelin olabileceği kadar doğru olmayacağı anlamına gelir.

Şekil 6-9: Aşırı uyumlu bir makine öğrenimi modeli

->>Major Types of Machine Learning Algorithms:Başlıca Makine Öğrenimi Algoritmaları Türleri

Şimdiye kadar, iki makine öğrenimi yöntemine değinerek, makine öğrenimini çok genel terimlerle tartıştım: lojistik regresyon ve en yakın komşu. logistic regression and k nearest neighbors

Bu bölümün geri kalanında, lojistik regresyonu, en yakın komşuları, karar ağaçlarını ve rastgele orman algoritmalarını daha ayrıntılı olarak inceliyor ve tartışıyoruz. Bu algoritmaları, güvenlik veri bilimi topluluğunda oldukça sık kullanıyoruz.Bu algoritmalar karmaşıktır, ancak bunların arkasındaki fikirler sezgisel ve anlaşılırdır.

İlk olarak, Şekil 6-10'da gösterilen her bir algoritmanın güçlü ve zayıf yönlerini keşfetmek için kullandığımız örnek veri kümelerine bakalım.

Bu veri setlerini örnek amaçlı oluşturdum. Sol tarafta, Şekil 6-7, 6-8 ve 6-9'da zaten kullandığım basit veri kümemiz var. Bu durumda, çizgi gibi basit bir geometrik yapı kullanarak siyah eğitim örneklerini (kötü amaçlı yazılım) gri eğitim örneklerinden (iyi huylu yazılım) ayırabiliriz.

Daha önce Şekil 6-6'da gösterdiğim sağdaki veri kümesi karmaşıktır çünkü basit bir satır kullanarak kötü amaçlı yazılımları iyi huylu yazılımlardan ayıramayız. Ancak verilerde hala net bir model var: bir karar sınırı oluşturmak için daha karmaşık yöntemler kullanmamız gerekiyor. Bu iki örnek veri kümesiyle farklı algoritmaların nasıl performans gösterdiğini görelim.

->Logistic Regression:

Daha önce öğrendiğiniz gibi, lojistik regresyon, eğitim kötü amaçlı yazılımınızı eğitim zararlı yazılımınızdan geometrik olarak ayıran bir çizgi, düzlem veya hiper düzlem (sağladığınız özelliğe bağlı olarak) oluşturan bir makine öğrenme algoritmasıdır. Yeni kötü amaçlı yazılımları tespit etmek için eğitimli modeli kullandığınızda, lojistik regresyon, daha önce görülmemiş bir ikili dosyanın kötü amaçlı mı yoksa zararlı mı olduğunu belirlemek için sınırın kötü amaçlı yazılım tarafında mı yoksa iyi huylu yazılım tarafında mı olduğunu kontrol eder.

Lojistik regresyonun bir sınırlaması, verileriniz sadece bir çizgi veya hiper düzlem kullanılarak ayrılamıyorsa, lojistik regresyon doğru çözüm değildir. Probleminiz için lojistik regresyon kullanıp kullanamayacağınız, verilerinize ve özelliklerinize bağlıdır. Örneğin, probleminizin kendi başına güçlü kötü niyetli (veya "iyi huylu olma") göstergeleri olan birçok bireysel özelliği varsa, lojistik regresyon kazanan bir yaklaşım olabilir. Verileriniz, bir dosyanın kötü amaçlı yazılım olduğuna karar vermek için özellikler arasında karmaşık ilişkiler kullanmanız gerekecek şekilde ise, k-en yakın komşular, karar ağaçları veya rastgele orman gibi başka bir yaklaşım daha mantıklı olabilir.

Lojistik regresyonun güçlü ve zayıf yönlerini göstermek için, Şekil 6-11'de gösterildiği gibi, iki örnek veri kümemizdeki lojistik regresyon performansına bakalım. Lojistik regresyonun, basit veri setimizde (solda) kötü amaçlı yazılım ile iyi huylu yazılım arasında çok etkili bir ayrım sağladığını görüyoruz. Bunun aksine, karmaşık veri setimizdeki lojistik regresyon performansı (sağda) etkili değildir. Bu durumda, lojistik regresyon algoritmasının kafası karışır, çünkü yalnızca doğrusal bir karar sınırını ifade edebilir. Her iki ikili türü de çizginin her iki tarafında görebilirsiniz ve gölgeli gri güven bantları verilere göre gerçekten bir anlam ifade etmez. Bu daha karmaşık veri kümesi için, daha geometrik yapıları ifade edebilen bir algoritma kullanmamız gerekir.

Şekil 6-11: Lojistik regresyon kullanılarak örnek veri setlerimiz aracılığıyla çizilen bir karar sınırı

->The Math Behind Logistic Regression:Lojistik Regresyonun Arkasındaki Matematik:

Şimdi lojistik regresyonun kötü amaçlı yazılım örneklerini nasıl algıladığının arkasındaki matematiğe bakalım. Liste 6-1, lojistik regresyon kullanarak bir ikili dosyanın kötü amaçlı yazılım olma olasılığını hesaplamak için Pythonic sözde kodunu gösterir.

def logistic\_regression(compressed\_data, suspicious\_calls, learned\_parameters): (1)

compressed\_data = compressed\_data \* learned\_parameters["compressed\_data\_weight"] (2)

suspicious\_calls = suspicious\_calls \* learned\_parameters["suspicious\_calls\_weight"]

score = compressed\_data + suspicious\_calls + bias (3)

return logistic\_function(score)

def logistic\_function(score): (4)

return 1/(1.0+math.e\*\*(-score))

Liste 6-1: Olasılığı hesaplamak için lojistik regresyon kullanan sözde kod

Bunun ne anlama geldiğini anlamak için kodun üzerinden geçelim. Önce logistic\_regression işlevini (1) ve parametrelerini tanımlarız.Parametreleri, sırasıyla sıkıştırılmış veri miktarını ve yaptığı şüpheli çağrıların sayısını temsil eden ikili (compressed\_data ve suspicious\_calls) özellikleridir ve learn\_parameters parametresi lojistik regresyon fonksiyonunun lojistik eğitimiyle öğrenilen öğelerini temsil eder. eğitim verilerinde regresyon modeli. Bu bölümde daha sonra parametrelerin nasıl öğrenildiğini tartışacağım; şimdilik, sadece eğitim verilerinden türetildiklerini kabul edin. Daha sonra, compressed\_data özelliğini (2) alıp onu compressed\_data\_weight parametresi ile çarpıyoruz. Bu ağırlık, lojistik gerileme işlevinin bu özelliğin ne kadar kötü amaçlı yazılım olduğunu düşündüğüne bağlı olarak özelliği yukarı veya aşağı ölçeklendirir. Ağırlığın da negatif olabileceğini unutmayın; bu, lojistik regresyon modelinin, özelliğin bir dosyanın zararsız olduğunun bir göstergesi olduğunu düşündüğünü gösterir.

Bunun altındaki satırda, suspicious\_calls parametresi için aynı adımı gerçekleştiriyoruz. Ardından, bu iki ağırlıklı özelliği (3) birlikte ekliyoruz, artı yanlılık parametresi olarak adlandırılan bir parametre ekliyoruz (ayrıca eğitim verilerinden öğrenildi). Özetle, ne kadar kötü niyetli olduğuna inandığımıza göre ölçeklenen compressed\_data özelliğini alıyoruz, şüpheli çağrılar özelliğini ekliyoruz, aynı zamanda kötü niyetli olduğuna inandığımız şekilde ölçekleniyor ve lojistiğin ne kadar şüpheli olduğunu gösteren regresyon modeli genel olarak dosyalardan olmamamız gerektiğini düşünüyor. Bu eklemelerin ve çarpmaların sonucu, belirli bir dosyanın kötü niyetli olma olasılığını gösteren bir puandır.

Son olarak, şüpheli puanımızı bir olasılığa dönüştürmek için logistic\_function (4) kullanırız. Şekil 6-12, bu işlevin nasıl çalıştığını görselleştirmektedir.

Burada, lojistik fonksiyon bir puan alır (x ekseninde gösterilir) ve bunu 0 ile 1 arasında sınırlanmış bir değere (olasılık) çevirir.

->How the Math Works:

Bu matematiğin pratikte nasıl çalıştığını görmek için Şekil 6-11'de gördüğünüz karar sınırlarına dönelim. Olasılığımızı nasıl hesapladığımızı hatırlayın:

logistic\_function(feature1\_weight \* feature1 + feature2\_weight\*feature2 + bias)

Örneğin, Şekil 6 11'de gösterilen özellik uzaylarının her noktasında ortaya çıkan olasılıkları aynı özellik ağırlıklarını ve önyargı parametresini kullanarak çizecek olsaydık, aynı şekilde gösterilen gölgeli bölgelerle sonuçlanırdık. model kötü niyetli ve iyi huylu örneklerin ne kadar güvenle yalan söylediğini "düşünür". O zaman 0,5'lik bir eşik ayarlasaydık (yüzde 50'den büyük olasılıkla dosyaların kötü niyetli olarak tanımlandığını hatırlayın), Şekil 6-11'deki satır karar sınırımız olarak görünecektir. Örnek kodumu denemenizi, bazı özellik ağırlıkları ve bir önyargı terimi eklemenizi ve kendiniz denemenizi tavsiye ederim.

Not:Lojistik regresyon, bizi yalnızca iki özelliği kullanmakla sınırlamaz. Gerçekte, genellikle lojistik regresyonlu puanlar veya yüzlerce, hatta binlerce özellik kullanırız. Ancak matematik değişmez: olasılığımızı herhangi bir sayıda özellik için aşağıdaki gibi hesaplarız:

logistic\_function(feature1 \* feature1\_weight + feature2 \* feature2\_weight +

feature3 \* feature3\_weight ... + bias)

Öyleyse lojistik regresyon, eğitim verilerine dayanarak karar sınırını doğru yere yerleştirmeyi tam olarak nasıl öğrenir? Gradyan inişi adı verilen yinelemeli, hesap tabanlı bir yaklaşım kullanır. Bu yaklaşımın ayrıntılarına bu kitapta girmeyeceğiz, ancak temel fikir, çizgi, düzlem veya hiper düzlemin (kullandığınız özelliklerin sayısına bağlı olarak) yinelemeli olarak ayarlanması ve böylece olasılığını en üst düzeye çıkarmasıdır. lojistik regresyon modeli, eğitim setindeki bir veri noktasının kötü amaçlı yazılım örneği mi yoksa iyi huylu yazılım örneği mi olduğu sorulduğunda yanıtı doğru alır.

Lojistik regresyon öğrenme algoritmasını, neyin kötü amaçlı yazılım ve iyi huylu yazılım oluşturduğuna dair daha basit veya daha karmaşık teoriler üretmeye yönlendirmek için lojistik regresyon modellerini eğitebilirsiniz. Bu eğitim yöntemleri bu kitabın kapsamı dışındadır, ancak bu yararlı yöntemleri öğrenmekle ilgileniyorsanız, sizi Google "lojistik regresyon ve düzenleme" işlemine ve bunların çevrimiçi açıklamalarını okumaya davet ediyorum.

->When to Use Logistic Regression:

Lojistik regresyon, diğer makine öğrenme algoritmalarına göre farklı avantaj ve dezavantajlara sahiptir. Lojistik regresyonun bir avantajı, lojistik regresyon modelinin neyin iyi huylu yazılım ve kötü amaçlı yazılım olduğunu düşündüğünü kolayca yorumlayabilmesidir. Örneğin, belirli bir lojistik regresyon modelini, özellik ağırlıklarına bakarak anlayabiliriz. Ağırlığı yüksek olan özellikler, modelin kötü niyetli olarak yorumladığı özelliklerdir. Negatif ağırlıklı özellikler, modelin iyi huylu yazılım olduğuna inandığı özelliklerdir. Lojistik regresyon oldukça basit bir yaklaşımdır ve üzerinde çalıştığınız veriler açık kötü niyetli göstergeler içerdiğinde iyi çalışabilir. Ancak veriler daha karmaşık olduğunda, lojistik regresyon genellikle başarısız olur.

Şimdi, çok daha karmaşık karar sınırlarını ifade edebilen başka bir basit makine öğrenimi yaklaşımını inceleyelim: en yakın komşular.

->K-Nearest Neighbors:

K-en yakın komşular, özellik alanındaki bir ikili dosyanın kötü niyetli diğer ikili dosyalara yakın olması durumunda kötü niyetli olduğu ve özellikleri onu iyi huylu ikili dosyalara yakın yerleştirirse zararsız olması gerektiği fikrine dayanan bir makine öğrenme algoritmasıdır. . Daha kesin olarak, bilinmeyen bir ikiliye en yakın k ikili dosyaların çoğu kötü niyetli ise, dosya kötü amaçlıdır. K'nin, bir örneğin iyi huylu veya kötü niyetli olup olmadığını belirlemede kaç komşunun dahil edilmesi gerektiğini düşündüğümüze bağlı olarak, kendimizi seçip tanımladığımız yakındaki komşuların sayısını temsil ettiğini unutmayın.

Gerçek dünyada bu sezgisel bir anlam ifade ediyor. Örneğin, hem basketbol oyuncularının hem de masa tenisi oyuncularının ağırlıklarına ve boylarına ilişkin bir veri kümesine sahipseniz, basketbol oyuncularının ağırlıkları ve boyları muhtemelen masa tenisi oyuncularının ölçümlerine olduğundan daha yakın olabilir. Benzer şekilde, bir güvenlik ortamında kötü amaçlı yazılımlar genellikle diğer kötü amaçlı yazılımlara benzer özelliklere sahip olur ve iyi huylu yazılımlar genellikle diğer iyi huylu yazılımlara benzer özelliklere sahip olur.

Aşağıdaki adımları kullanarak bir ikili dosyanın kötü niyetli mi yoksa zararsız mı olduğunu hesaplamak için bu fikri en yakın komşu algoritmasına çevirebiliriz:

1. İkilinin özelliklerini çıkarın ve özellik uzayında ona en yakın olan k örneğini bulun.

2. En yakın kötü niyetli komşuların yüzdesini elde etmek için örneğe yakın olan kötü amaçlı yazılım örneklerinin sayısını k'ye bölün.

3. Yeterli sayıda örnek kötü amaçlıysa, örneği kötü amaçlı olarak tanımlayın.

Şekil 6-13 k-en yakın komşu algoritmasının yüksek düzeyde nasıl çalıştığını göstermektedir.

Şekil 6-13: Daha önce görülmeyen kötü amaçlı yazılımları tespit etmek için en yakın komşunun nasıl kullanılabileceğinin bir örneği

Sol üstte bir dizi kötü amaçlı yazılım eğitimi örneği ve sağ altta bir dizi iyi huylu yazılım örneği görüyoruz. Ayrıca en yakın üç komşusuna bağlı yeni, bilinmeyen bir ikili görüyoruz. Bu durumda, k'yi 3'e ayarladık, yani bilinmeyen ikili dosyalara en yakın üç komşuya bakıyoruz. En yakın komşuların üçü de kötü niyetli olduğu için, bu yeni ikiliyi kötü niyetli olarak sınıflandıracağız.

->The Math Behind K-Nearest Neighbors:K-En Yakın Komşuların Arkasındaki Matematik

Şimdi yeni, bilinmeyen ikili dosyaların özellikleri ile eğitim setindeki örnekler arasındaki mesafeyi hesaplamamıza izin veren matematiği tartışalım. Bunu yapmak için, bize yeni örneğimiz ile eğitim setindeki örnekler arasındaki mesafeyi söyleyen bir mesafe işlevi kullanıyoruz. En yaygın uzaklık işlevi, özellik uzayımızdaki iki nokta arasındaki en kısa yolun uzunluğu olan Öklid mesafesidir. Liste 6-2, örnek iki boyutlu özellik uzayımızda Öklid mesafesi için sözde kodu gösterir.

import math

def euclidean\_distance(compression1,suspicious\_calls1, compression2, suspicious\_calls2): (1)

comp\_distance = (compression1-compression2)\*\*2 (2)

call\_distance = (suspicious\_calls1-suspicious\_calls2)\*\*2 (3)

return math.sqrt(comp\_distance + call\_distance) (4)

Bu koddaki matematiğin nasıl çalıştığını inceleyelim. Liste 6-2, bir çift örnek alır ve özellikleri arasındaki farklara göre aralarındaki mesafeyi hesaplar. İlk olarak, arayan kişi ikili dosyaların (1) özelliklerinden geçer, burada sıkıştırma1 ilk örneğin sıkıştırma özelliği, suspicious\_calls1 ilk örneğin suspicious\_calls özelliği, sıkıştırma2 ikinci örneğin sıkıştırma özelliği ve suspicious\_calls2 ise şüpheli ikinci örneğin aramalar özelliği.

Daha sonra her örneğin (2) sıkıştırma özelliklerinin karesi alınmış farkını hesaplıyoruz ve her örneğin (2) şüpheli çağrı özelliği arasındaki karesi alınmış farkı hesaplıyoruz. Kare mesafeyi kullanma nedenimizi kapsamayacağız, ancak ortaya çıkan farkın her zaman pozitif olduğunu unutmayın. Son olarak, iki öznitelik vektörü arasındaki Öklid uzaklığı olan iki farklılığın karekökünü hesaplayıp arayana geri döndürüyoruz (4). Örnekler arasındaki mesafeleri hesaplamanın başka yolları olsa da, Öklid mesafesi en yaygın k en yakın komşu algoritmasıyla kullanılır ve güvenlik veri bilimi problemlerinde iyi çalışır.

->Choosing the Number of Neighbors That Vote:Oy Verecek Komşu Sayısını Seçmek:

Şimdi bu bölümde kullandığımız örnek veri kümeleri için k-en yakın komşu algoritmasının ürettiği karar sınırları ve olasılık türlerine bakalım. Şekil 6-14'te k'yi 5'e ayarladım, böylece en yakın beş komşunun "oy kullanmasına" izin verdim.

Şekil 6-14: k 5 olarak ayarlandığında k-en yakın komşular tarafından oluşturulan karar sınırları

Ancak Şekil 6-15'te k'yi 50'ye ayarladım ve en yakın 50 komşunun "oy kullanmasına" izin verdim.

Şekil 6-15: k 50'ye ayarlandığında k en yakın komşuları tarafından oluşturulan karar sınırları

Oy veren komşuların sayısına bağlı olarak modeller arasındaki çarpıcı farklılığa dikkat edin. Şekil 6-14'teki model, her iki veri seti için de aşırı ve karmaşık bir karar sınırını göstermektedir; bu sınır, aykırı değerlerin etrafına yerel karar sınırlarını çizmesi anlamında aşırı uygundur, ancak basit, genel eğilimleri yakalayamadığı için yetersizdir. Bunun aksine, Şekil 6-15'teki model her iki veri kümesine de çok uygundur, çünkü aykırı değerler tarafından dikkati dağılmaz ve genel eğilimleri net bir şekilde tanımlar. Gördüğünüz gibi, k-en yakın komşular, lojistik regresyondan çok daha karmaşık bir karar sınırı oluşturabilir. Bir örneğin kötü huylu veya kötü huylu olup olmadığına oy verecek komşu sayısı olan k'yi değiştirerek hem aşırı hem de yetersiz uyuma karşı koruma sağlamak için bu sınırın karmaşıklığını kontrol edebiliriz. Şekil 6-11'deki lojistik regresyon modeli bunu tamamen yanlış anlasa da, k-en yakın komşular, özellikle 50 komşunun oy kullanmasına izin verdiğimizde, kötü amaçlı yazılımı iyi huylu yazılımdan ayırmada başarılı oluyor. K-en yakın komşular doğrusal bir yapı tarafından sınırlandırılmadığından ve bir karar vermek için her noktanın en yakın komşusuna baktığından, keyfi şekillerle karar sınırları oluşturabilir, böylece karmaşık veri kümelerini çok daha etkili bir şekilde modelleyebilir.

->When to Use K-Nearest Neighbors:

K-en yakın komşular, özelliklerin şüpheli kavramla net bir şekilde eşleşmediği, ancak kötü niyetli örneklere yakınlığın güçlü bir kötü niyet göstergesi olduğu verileriniz olduğunda göz önünde bulundurmanız gereken iyi bir algoritmadır. Örneğin, kötü amaçlı yazılımları, kodu paylaşan aileler olarak sınıflandırmaya çalışıyorsanız, k-en yakın komşular denemek için iyi bir algoritma olabilir, çünkü bir kötü amaçlı yazılım örneğini, özellikleri bir kötü amaçlı yazılımın bilinen üyelerine benziyorsa bir aile içinde sınıflandırmak isteyebilirsiniz. verilen aile.

K-en yakın komşuları kullanmanın bir başka nedeni de, neden belirli bir sınıflandırma kararı verdiğine dair net açıklamalar sağlamasıdır. Başka bir deyişle, algoritmanın onu neden kötü amaçlı yazılım veya zararlı yazılım olarak sınıflandırdığını anlamak için örnekler ve bilinmeyen bir örnek arasındaki benzerlikleri belirlemek ve karşılaştırmak kolaydır.

->Decision Trees: Karar Ağaçları:

Karar ağaçları, algılama problemlerini çözmek için sıklıkla kullanılan bir başka makine öğrenimi yöntemidir. Karar ağaçları, Twenty Questions oyununa benzer şekilde, belirli bir ikili dosyanın kötü amaçlı yazılım olup olmadığına karar vermek için bir eğitim süreci aracılığıyla otomatik olarak bir dizi soru oluşturur. Şekil 6-16, bu bölümde kullanmakta olduğumuz basit veri kümesine göre eğiterek otomatik olarak oluşturduğum bir karar ağacını göstermektedir. Ağaçtaki mantığın akışını takip edelim.

Karar ağacı akışı, yeni, daha önce görülmemiş bir ikili dosyadan çıkardığımız özellikleri ağaca girdiğimizde başlar. Ardından ağaç, bu ikilinin özellikleriyle ilgili sorulacak soru dizisini tanımlar. Ağacın tepesindeki düğüm dediğimiz kutu ilk soruyu sorar: Ağaçtaki şüpheli çağrıların sayısı 40.111'den az mı yoksa eşit mi? Karar ağacının burada bir kayan nokta sayısı kullandığına dikkat edin, çünkü her ikilideki şüpheli çağrıların sayısını 0 ile 100 arasındaki bir aralığa normalleştirdik. Cevap "evet" ise, başka bir soru soruyoruz: sıkıştırılmışların yüzdesidir. dosyadaki veriler 37.254'ten küçük veya eşit mi? Cevap "evet" ise, sonraki soruya geçiyoruz: ikili dosyadaki şüpheli çağrıların sayısı 33.836'dan az mı yoksa eşit mi? Cevap “evet” ise, karar ağacının sonuna ulaşıyoruz. Bu noktada, ikili dosyanın kötü amaçlı yazılım olma olasılığı yüzde 0'dır.

Şekil 6-17, bu karar ağacının geometrik bir yorumunu göstermektedir.

Şekil 6-17: Basit veri seti örneğimiz için bir karar ağacı tarafından oluşturulan karar sınırı

Burada gölgeli bölgeler, karar ağacının örneklerin kötü niyetli olduğunu düşündüğü yeri gösterir. Daha açık bölgeler, karar ağacının numunelerin iyi huylu olduğunu düşündüğü yerleri gösterir. Şekil 6-16'daki soru ve cevap dizileri tarafından belirlenen olasılıklar, Şekil 6-17'deki gölgeli bölgelerdekilere karşılık gelmelidir.

->Choosing a Good Root Node:İyi Bir Kök Düğüm Seçmek

Öyleyse, eğitim verilerinden buna benzer bir karar ağacı oluşturmak için bir makine öğrenimi algoritmasını nasıl kullanırız? Temel fikir, karar ağacının kök düğüm adı verilen bir ilk soruyla başlamasıdır. En iyi kök düğüm, bir türdeki tüm örnekler olmasa bile çoğu için "evet" yanıtı ve diğer türün tüm örnekleri olmasa da çoğu için "hayır" yanıtı aldığımız olandır. Örneğin, Şekil 6 16'da, kök düğüm sorusu, daha önce görülmeyen bir ikili dosyanın 40.111 veya daha az çağrısı olup olmadığını sorar. (İkili dosya başına çağrı sayısının 0 ila 100 ölçeğine normalleştirildiğini ve kayan nokta değerlerini geçerli kıldığını unutmayın.) Şekil 6-17'deki dikey çizgiden de görebileceğiniz gibi, iyi huylu verilerin çoğu bu sayıdan daha azdır. , kötü amaçlı yazılım verilerinin çoğunda bu sayıdan fazla şüpheli arama bulunur, bu da bunu sormak için iyi bir başlangıç sorusu yapar.

->Picking Follow-Up Questions:Takip Soruları Toplama

Bir kök düğüm seçtikten sonra, kök düğümü seçmek için kullandığımıza benzer bir yöntem kullanarak sonraki soruları seçin. Örneğin, kök düğüm örnekleri iki gruba ayırmamıza izin verdi: 40.111 şüpheli çağrıya eşit veya daha az olan bir grup (negatif özellik alanı) ve 40.111'den fazla şüpheli çağrıya sahip olan bir grup (pozitif özellik alanı). Bir sonraki soruyu seçmek için, özellik alanının her bir alanındaki örnekleri kötü niyetli ve zararsız eğitim örneklerine daha fazla ayıracak sorulara ihtiyacımız var.

Bunu, karar ağacının Şekil 6-16 ve 6 17'de yapılandırıldığı şekilde görebiliriz. Örneğin, Şekil 6-16, ikili dosyaların yaptığı şüpheli çağrıların sayısı hakkında bir başlangıç ​​"kök" sorusunu sorduktan sonra, sorduğumuzu gösterir. Sıkıştırılmış veri ikili dosyalarının ne kadar olduğuna dair sorular. Şekil 6-17, bunu neden verilere dayanarak yaptığımızı göstermektedir: Şüpheli işlev çağrıları hakkındaki ilk sorumuzu sorduktan sonra, çoğu kötü amaçlı yazılımı arsadaki çoğu iyi huylu yazılımdan ayıran kaba bir karar sınırına sahibiz. Takip soruları sorarak karar sınırını nasıl daha da iyileştirebiliriz? Karar sınırımızı daraltacak sorulacak bir sonraki en iyi sorunun ikili dosyalardaki sıkıştırılmış veri miktarı hakkında olacağı görsel olarak açıktır.

->When to Stop Asking Questions:Soru Sormayı Ne Zaman Bırakmalısınız?

Karar ağacı oluşturma sürecimizin bir noktasında, karar ağacının ne zaman soru sormayı bırakması gerektiğine karar vermemiz ve cevabımız hakkındaki kesinliğimiz temelinde bir ikili dosyanın zararsız mı yoksa kötü niyetli mi olduğunu belirlememiz gerekir. Bunun bir yolu, karar ağacımızın sorabileceği soruların sayısını veya derinliğini (herhangi bir ikili dosyadan sorabileceğimiz maksimum soru sayısı) sınırlamaktır. Bir diğeri, eğitim setimizdeki her örneğin ağacın yapısına bağlı olarak kötü amaçlı yazılım veya zararlı yazılım olup olmadığından kesinlikle emin olana kadar karar ağacının büyümeye devam etmesine izin vermektir. Ağacın boyutunu sınırlamanın avantajı, eğer ağaç daha basitse, yanıtı doğru alma şansımızın artmasıdır (Occam'ın usturasını düşünün - teori ne kadar basitse o kadar iyidir). Diğer bir deyişle, küçük tutarsak karar ağacının eğitim verilerini aşma şansı daha azdır.

Tersine, eğitim verilerini yetersiz kılıyorsak ağacın maksimum boyuta büyümesine izin vermek faydalı olabilir. Örneğin, ağacın daha da büyümesine izin vermek, karar sınırının karmaşıklığını artıracaktır; bu, yetersiz uyuyorsak bunu yapmak isteriz. Genel olarak, makine öğrenimi uygulayıcıları genellikle birden fazla derinlik dener veya daha önce görülmeyen ikili dosyalar üzerinde maksimum derinliğe izin vererek bu işlemi en doğru sonuçları elde edene kadar tekrarlar.

->Using Pseudocode to Explore Decision Tree Generation Algorithms:Karar Ağacı Oluşturma Algoritmalarını Keşfetmek için Sözde Kod Kullanma

Şimdi otomatik bir karar ağacı oluşturma algoritmasını inceleyelim. Bu algoritmanın arkasındaki temel fikrin, eğitim örneklerinin kötü niyetli mi yoksa zararsız mı olduğuna dair kesinliğimizi en iyi artıran soruyu bularak ve ardından kesinliğimizi daha da artıracak sonraki soruları bularak ağaçta kök düğümü oluşturmak olduğunu öğrendiniz. Algoritma, eğitim örnekleri hakkındaki kesinliği önceden belirlediğimiz bazı eşikleri aştığında soru sormayı bırakmalı ve bir karar vermelidir.

Programlı olarak bunu yinelemeli olarak yapabiliriz. Liste 6-3'teki Python benzeri sözde kod, basitleştirilmiş formda bir karar ağacı oluşturmak için tüm süreci gösterir.

tree = Tree()

def add\_question(training\_examples):

(1) question = pick\_best\_question(training\_examples)

(2)uncertainty\_yes,yes\_samples=ask\_question(question,training\_examples,"yes")

(3)uncertainty\_no,no\_samples=ask\_question(question,training\_examples,"no")

(4)if not uncertainty\_yes < MIN\_UNCERTAINTY:

add\_question(yes\_samples)

(5)if not uncertainty\_no < MIN\_UNCERTAINTY:

add\_question(no\_samples)

(6)add\_question(training\_examples)

6-3 Listeleme: Karar ağacı algoritması oluşturmak için sözde kod

Sözde kod, bir karar ağacına özyinelemeli olarak sorular ekler, kök düğümden başlayarak ve algoritma, karar ağacının yeni bir dosyanın iyi huylu veya kötü niyetli olup olmadığına dair son derece kesin bir yanıt sağlayabileceğinden emin olana kadar aşağı doğru çalışır.

Ağacı oluşturmaya başladığımızda, kök düğümümüzü (1) seçmek için pick\_best\_question () kullanırız (şimdilik, bu işlevin nasıl çalıştığı konusunda endişelenmeyin). Ardından, bu ilk soruya (2) cevabının “evet” olduğu eğitim örnekleri hakkında şu anda ne kadar belirsizliğe sahip olduğumuza bakarız. Bu, bu örnekler hakkında soru sormaya devam etmemiz gerekip gerekmediğine veya durup, örneklerin kötü niyetli mi yoksa zararsız mı olduğunu tahmin etmemize karar vermemize yardımcı olacaktır. İlk soru (3) için “hayır” cevabını verdiğimiz örnekler için de aynısını yapıyoruz. Daha sonra, "evet" (belirsizlik\_yes) olarak yanıtladığımız örneklerle ilgili sahip olduğumuz belirsizliğin, bunların kötü niyetli mi yoksa zararsız mı olduğuna karar vermek için yeterince düşük olup olmadığını kontrol ederiz (4). Bu noktada kötü niyetli mi yoksa zararsız mı olduklarını belirleyebilirsek, ek soru sormayız. Ancak yapamazsak, yes\_samples kullanarak veya girdi olarak "evet" olarak yanıtladığımız örnek sayısını kullanarak add\_question () 'u tekrar çağırırız. Bu, kendisini çağıran bir işlev olan klasik bir özyineleme örneğidir. Kök düğüm için gerçekleştirdiğimiz aynı işlemi, eğitim örneklerinin bir alt kümesiyle tekrarlamak için özyinelemeyi kullanıyoruz. Sonraki if ifadesi "hayır" örneklerimiz için aynı şeyi yapar (5). Son olarak, eğitim örneklerimizde (6) karar ağacı oluşturma işlevimizi adlandırıyoruz.

Pick\_best\_question () tam olarak nasıl çalışır, bu kitabın kapsamı dışında kalan matematiği içerir, ancak fikir basittir. Karar ağacı oluşturma sürecinin herhangi bir noktasında en iyi soruyu seçmek için, hala belirsiz olduğumuz eğitim örneklerine bakarız, onlar hakkında sorabileceğimiz tüm soruları sıralar ve ardından belirsizliğimizi en iyi azaltan soruyu seçeriz örneklerin kötü amaçlı yazılım veya iyi huylu yazılım olup olmadığı. Belirsizlikteki bu azalmayı bilgi kazancı adı verilen istatistiksel bir ölçüm kullanarak ölçüyoruz. En iyi soruyu seçmenin bu basit yöntemi şaşırtıcı derecede iyi çalışıyor.

Not:Bu, gerçek dünya, karar ağacı oluşturma, makine öğrenimi algoritmalarının nasıl çalıştığına dair basitleştirilmiş bir örnektir. Belirli bir sorunun bir dosyanın kötü olup olmadığına dair kesinliğimizi ne kadar artırdığını hesaplamak için gereken matematiği atladım.

Şimdi bu bölümde kullanmakta olduğumuz iki örnek veri kümesindeki karar ağaçlarının davranışına bakalım. Şekil 6-18, bir karar ağacı algılayıcısı tarafından öğrenilen karar sınırını gösterir.

Şekil 6-18: Karar ağacı yaklaşımıyla üretilen örnek veri kümelerimiz için karar sınırları

Bu durumda, ağaçlar için bir maksimum derinlik ayarlamak yerine, her eğitim örneğinin doğru şekilde sınıflandırılması için eğitim verilerine göre yanlış pozitif veya yanlış negatiflerin olmadığı noktaya kadar büyümelerine izin veriyoruz.

Eğri veya çapraz bir çizginin daha uygun olabileceği net ve açık görünse bile, karar ağaçlarının özellik uzayında yalnızca yatay ve dikey çizgiler çizebileceğine dikkat edin. Bunun nedeni, karar ağaçlarının yalnızca bireysel özellikler üzerinde basit koşulları ifade etmemize izin vermesidir (büyüktür veya eşittir ve eşittir veya eşittir), bu da her zaman yatay veya dikey çizgilere yol açar. Ayrıca, bu örneklerdeki karar ağaçlarının iyi huylu yazılımı kötü amaçlı yazılımdan ayırmada başarılı olmasına rağmen, karar sınırlarının son derece düzensiz göründüğünü ve garip yapılara sahip olduğunu da görebilirsiniz. Örneğin, kötü amaçlı yazılım bölgesi, garip yollarla iyi huylu yazılım bölgesine genişler ve bunun tersi de geçerlidir. Olumlu tarafı, karar ağacı, karmaşık veri kümesi için bir karar sınırı oluşturmada lojistik regresyondan çok daha iyidir.

Şimdi Şekil 6-18'deki karar ağaçlarını Şekil 6-19'daki karar ağacı modelleriyle karşılaştıralım.

Şekil 6-19: Sınırlı derinlikli bir karar ağacı tarafından üretilen örnek veri kümelerimiz için karar sınırları

Şekil 6-19'daki karar ağaçları, ağaç derinliğini beş düğümle sınırlamam dışında, Şekil 6-18 için kullanılan aynı karar ağacı oluşturma algoritmasını kullanır. Bu, herhangi bir ikili dosya için, özellikleriyle ilgili en fazla beş soru sorabileceğim anlamına gelir.

Sonuç dramatik. Şekil 6-18'de gösterilen karar ağacı modelleri, aşırı değerlere odaklanarak ve genel eğilimi yakalayamayan aşırı karmaşık sınırlar çizerek açıkça aşırı uyumluyken, Şekil 6-19'daki karar ağaçları, genel bir model belirleyerek verilere çok daha zarif bir şekilde uyuyor aykırı değerlere odaklanmadan her iki veri kümesinde de (bir istisna dışında, basit veri kümesinin sağ üst alanındaki daha ince karar bölgesi). Gördüğünüz gibi, iyi bir maksimum karar ağacı derinliği seçmek, karar ağacı tabanlı makine öğrenimi dedektörünüz üzerinde büyük bir etkiye sahip olabilir.

->When to Use Decision Trees:

Karar ağaçları anlamlı ve basit olduğundan, basit evet veya hayır sorularına dayalı olarak hem basit hem de oldukça düzensiz sınırları öğrenebilirler. Kötü amaçlı yazılıma karşı iyi huylu yazılımın ne olduğu konusundaki teorilerinin ne kadar basit veya karmaşık olması gerektiğini kontrol etmek için maksimum derinliği de ayarlayabiliriz.

Ne yazık ki, karar ağaçlarının dezavantajı, genellikle çok doğru modellerle sonuçlanmamalarıdır. Bunun nedeni karmaşıktır, ancak karar ağaçlarının, daha önce görülmemiş örneklere çok iyi genelleştirecek şekilde eğitim verilerine uymayan pürüzlü karar sınırlarını ifade etmesiyle ilgilidir.

Benzer şekilde, karar ağaçları genellikle karar sınırları etrafında doğru olasılıkları öğrenmezler. Bunu Şekil 6-19'da karar sınırı etrafındaki gölgeli bölgeleri inceleyerek görebiliriz. Bozulma doğal veya aşamalı değildir ve olması gereken bölgelerde, yani kötü amaçlı yazılım ve iyi huylu yazılım örneklerinin çakıştığı alanlarda meydana gelmez.

Ardından, çok daha iyi sonuçlar elde etmek için birden çok karar ağacını birleştiren rastgele orman yaklaşımını tartışacağım.

->Random Forest:

Güvenlik topluluğu kötü amaçlı yazılım tespiti için büyük ölçüde karar ağaçlarına güvenmesine rağmen, bunları neredeyse hiçbir zaman tek tek kullanmazlar. Bunun yerine, rastgele orman adı verilen bir yaklaşımla tespit yapmak için yüzlerce veya binlerce karar ağacı birlikte kullanılır. Tek bir karar ağacı eğitmek yerine, birçok, genellikle yüz veya daha fazla kişiyi eğitiyoruz, ancak her karar ağacını farklı şekilde eğitiyoruz, böylece veriler üzerinde farklı bir bakış açısına sahip oluyoruz. Son olarak, yeni bir ikili dosyanın kötü niyetli mi yoksa zararsız mı olduğuna karar vermek için karar ağaçlarının oy kullanmasına izin veriyoruz. Bir ikili dosyanın kötü amaçlı yazılım olma olasılığı, pozitif oy sayısının toplam ağaç sayısına bölünmesiyle elde edilir.

Tabii ki, eğer tüm karar ağaçları aynı ise, hepsi aynı şekilde oy kullanır ve rastgele orman, bireysel karar ağaçlarının sonuçlarını basitçe kopyalar. Bunu ele almak için, karar ağaçlarının kötü amaçlı ve zararlı yazılımları neyin oluşturduğuna dair farklı bakış açılarına sahip olmasını istiyoruz ve bu çeşitliliği karar ağaçları koleksiyonumuza dahil etmek için aşağıda tartışacağım iki yöntem kullanıyoruz. Çeşitliliği teşvik ederek, modelimizde tipik olarak daha doğru bir modelle sonuçlanan bir "kalabalıkların bilgeliği wisdom of crowds" dinamiği oluşturuyoruz.

Rastgele bir orman algoritması oluşturmak için aşağıdaki adımları kullanıyoruz:

1. Eğitim: Oluşturmayı planladığımız sayıdaki her ağaç için (genellikle 100 veya daha fazla)

• Eğitim setimizden bazı eğitim örneklerini rastgele örnekleyin.

• Rastgele örnekten bir karar ağacı oluşturun.

• Oluşturduğumuz her ağaç için, her "soru sormayı" düşündüğümüzde, yalnızca birkaç özellikten oluşan bir soru sormayı düşünün ve diğer özellikleri göz ardı edin.

2. Daha önce görülmemiş bir ikili dosyada algılama

• İkilideki her bir ağaç için algılamayı çalıştırın.

• İkilinin kötü amaçlı yazılım olup olmadığına "evet" oyu veren ağaçların sayısına göre karar verin.

Bunu daha ayrıntılı olarak anlamak için, Şekil 6-20'de gösterildiği gibi, iki örnek veri setimizde rastgele orman yaklaşımı tarafından üretilen sonuçları inceleyelim. Bu sonuçlar 100 karar ağacı kullanılarak oluşturulmuştur.

Şekil 6-20: Rastgele orman yaklaşımı kullanılarak oluşturulan karar sınırları

Şekil 6-18 ve 6-19'da gösterilen bağımsız karar ağacı sonuçlarının aksine, rastgele orman, hem basit hem de karmaşık veri kümeleri için ayrı karar ağaçlarından çok daha yumuşak ve sezgisel karar sınırları ifade edebilir. Gerçekten de, rastgele orman modeli, eğitim veri setine pürüzlü kenarlar olmadan çok temiz bir şekilde uyar; model, her iki veri kümesi için neyin "kötü niyetli veya iyi huylu" olduğu hakkında iyi teoriler öğrenmiş gibi görünüyor.

Ek olarak, gölgeli bölgeler sezgiseldir. Örneğin, iyi huylu veya kötü niyetli örneklerden ne kadar uzaklaşırsanız, rasgele orman örneklerin kötü niyetli mi yoksa zararsız mı olduğu konusunda o kadar az kesinlik kazanır. Bu, rastgele ormanın önceden görülmemiş ikili dosyalar üzerindeki performansı için iyi bir işarettir.

Aslında, bir sonraki bölümde göreceğiniz gibi, rastgele orman, bu bölümde tartışılan tüm yaklaşımların daha önce görülmemiş ikili dosyalarında en iyi performans gösteren modeldir.

Rastgele ormanın bireysel karar ağaçlarına kıyasla neden bu kadar net karar sınırları çizdiğini anlamak için 100 karar ağacının ne yaptığını düşünelim. Her ağaç, eğitim verilerinin yalnızca üçte ikisini görür ve hangi soruyu soracağına karar verdiğinde yalnızca rastgele seçilen bir özelliği dikkate alır. Bu, perde arkasında, örneklerde (ve gölgeli bölgelerde) nihai karar sınırlarını oluşturmak için ortalaması alınan 100 farklı karar sınırımız olduğu anlamına gelir. Bu "kalabalıkların bilgeliği" dinamiği, verilerdeki eğilimleri bireysel karar ağaçlarının yapabileceğinden çok daha karmaşık bir şekilde tanımlayabilen toplu bir fikir yaratır.

-> ÖZET:

Bu bölümde, makine öğrenimine dayalı kötü amaçlı yazılım tespitine üst düzey bir girişin yanı sıra makine öğrenimine yönelik dört ana yaklaşım var: lojistik regresyon, en yakın komşular, karar ağaçları ve rastgele ormanlar. Makine öğrenimi tabanlı algılama sistemleri, algılama imzaları yazma işini otomatik hale getirebilir ve pratikte genellikle özel yazılı imzalardan daha iyi performans gösterir.

Aşağıdaki bölümlerde, size bu yaklaşımların gerçek dünyadaki kötü amaçlı yazılım algılama sorunlarında nasıl performans gösterdiğini göstereceğim. Özellikle, dosyaları kötü niyetli veya zararsız olarak doğru bir şekilde sınıflandırmak için makine öğrenimi algılayıcıları oluşturmak için açık kaynak, makine öğrenimi yazılımını nasıl kullanacağınızı ve algılayıcılarınızın daha önce görülmemiş ikili dosyalar üzerindeki performansını değerlendirmek için temel istatistikleri nasıl kullanacağınızı öğreneceksiniz.