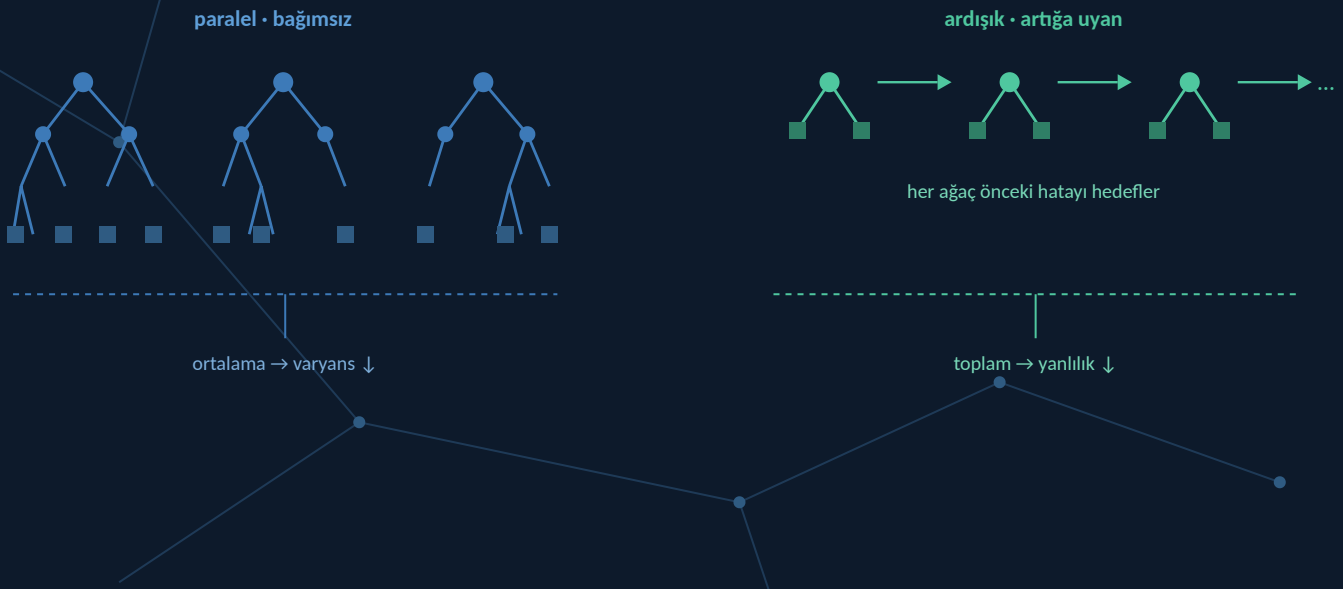
BCDataWorks

AĞAÇ TOPLULUKLARI

Random Forest XGBoost

Bagging ile boosting'i gerçekten ayıran tek karar, parametre karşılıkları ve kredi riski analitiğinde hangisinin ne zaman kazandığı.



· Tasarım farkları · Parametre eşlemesi · Kredi riski kıyası

Bu doküman ne anlatıyor?

İkisi de karar ağacı topluluğudur ve ikisi de aynı CART bölünmesini kullanır. Fark tek bir yerdedir: ağaçlar **birbirinden bağımsız mı** yoksa **birbirinin hatası üzerine mi** kurulur. Bu tek karar, ağaç sayısından derinliğe, erken durdurmadan kalibrasyona kadar her şeyi değiştirir.

İki cümlede fark

■ **Random Forest** — birbirinden bağımsız, derin ve gürültülü ağaçları paralel büyütür; ortalamayla **varyansı** düşürür. Varsayılanları güvenlidir, ayar yükü düşüktür, aşırı öğrenmeye karşı doğal olarak dirençlidir.

● **XGBoost** — sığ ağaçları ardışık büyütür; her ağaç önceki modelin gradyanına uyar ve **yanlılığı** düşürür. Doğru ayarlandığında tabular veride tavanı daha yüksektir, yanlış ayarlandığında sessizce ezberler.

İçindekiler

Sürüm ve ölçüm notu

Anlatım **scikit-learn 1.4+** (RandomForestClassifier) ve **XGBoost 3.x** sürüm aileleri esas alınarak yazılmıştır. Eksik değer desteği ve monotonluk kısıtı gibi bazı davranışlar scikit-learn'de görece yenidir; kritik bir karar vermeden önce kullandığın sürümün dokümantasyonundan doğrula.

Şekillerdeki süre, bellek ve performans sayıları **temsilîdir** — tipik davranışın yönünü göstermek içindir, benchmark sonucu değildir. Kendi verinde ölçmeden karar verme; 15. bölüm bunun protokolünü verir.

01 İki yaklaşım tek bakışta

İkisi de karar ağacı topluluğudur. Farkları, çözmek için tasarlandıkları problemden gelir: biri tek bir ağacın **kararsızlığını**, diğeri tek bir ağacın **yetersizliğini** hedefler.

Aynı zayıflığın iki farklı okuması

Random Forest · Breiman, 2001

Sorun: tek ağaç kararsızdır

Veriyi biraz değiştir, ağaç tamamen değişir.
Yani varyansı yüksektir.

Çözüm: bootstrap + rastgele değişken alt kümesi, sonra ortalama al.

VS

XGBoost · Chen & Guestrin, 2016

Sorun: sığ ağaç yetersizdir

Tek bir sığ ağaç sinyalin çoğunu kaçırır.
Yani yanlılığı yüksektir.

Çözüm: kalan hatayı hedefleyen yeni bir ağaç ekle, üstüne topla.

İkisi de doğrudur — biri gürültüyü söndürür, diğeri sinyali kovalar.

Şekil 1. Aynı ağaç zayıflığına verilen iki farklı cevap.

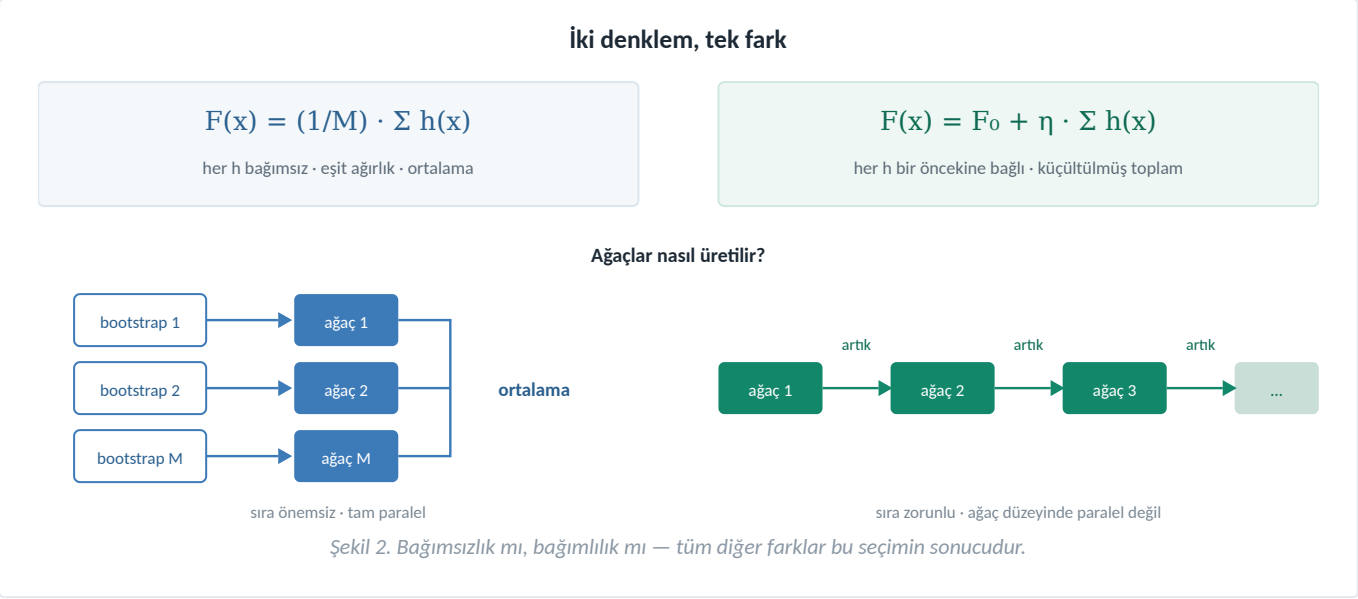
	Random Forest	XGBoost
Geliştirici / yıl	Breiman, 2001 (sklearn, ranger, H2O)	DMLC, 2014–2016
Topluluk mantığı	Bagging — bağımsız ağaçların oyu/ortalaması	Boosting — ardışık ağaçların toplamı
Tek ağacın rolü	Güçlü öğrenici (derin, tam büyümüş)	Zayıf öğrenici (sığ, derinlik 3–6)
Ağaçlar arası ilişki	Bağımsız — paralel eğitilebilir	Bağımlı — sıra bozulamaz
Ana rastgelelik	Bootstrap + `max_features`	Öğrenme oranı + örnekleme (opsiyonel)
En güçlü yanı	Varsayılanların güvenliği; gürültüye ve ayar hatasına dayanıklılık	Aynı veriden çıkarılabilecek en yüksek ayrım gücü
En zayıf yanı	Tavanı görece düşük; model boyutu büyük	Ayar ve erken durdurma disiplini gerektirir
Ayar yükü	Düşük — 2–3 düğme	Yüksek — 8–10 düğme, birbirine bağlı

Önemli çerçeve

"Hangisi daha iyi?" sorusunun cevabı literatürde nettir: iyi ayarlanmış boosting, tabular veride Random Forest'ı genellikle geçer. Fakat **pratikte sorulması gereken soru şudur:** bu ekibin ayar, validasyon ve izleme disiplini, XGBoost'un sunduğu fazladan ayrım gücünü güvenli biçimde toplayabilecek kadar sıkı mı? Değilse iyi kurulmuş bir Random Forest, kötü ayarlanmış bir XGBoost'tan iyidir.

02 Ortak temel, ayrışan tek karar

Alt katman aynıdır: ikisi de CART ağacı büyütür, bölünme adaylarını aynı mantıkla arar, ölçeklemeye ihtiyaç duymaz. Ayrışma tek bir denklemde görünür.



Şekil 2. Bağımsızlık mı, bağımlılık mı — tüm diğer farklar bu seçimin sonucudur.

Aynı olanlar

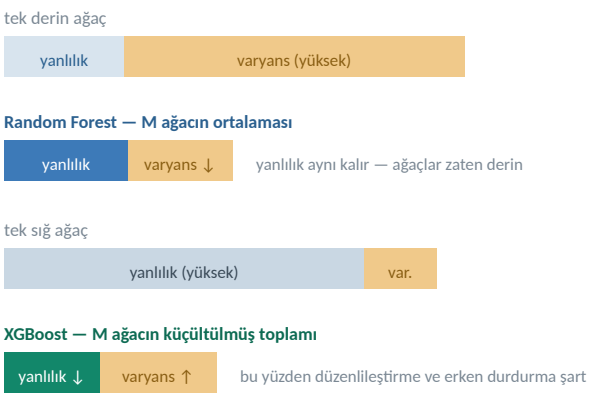
· CART bölünme mantığı; eşik aramanın maliyeti satır × değişken × aday eşik ile büyür.
· Değişkenlerin ölçeklenmesine veya normalleştirilmesine ihtiyaç duymamaları.
· Monotonik dönüşümlere (log, karekök) duyarsız olmaları — sıralama değişmedikçe sonuç değişmez.
· Satır ve kolon örneklemesiyle varyans azaltabilmeleri.
· TreeSHAP ile aynı yöntemle açıklanabilmeleri.
· Aykırı değere görece dayanıklılık: bölünme sıralamaya bakar, uzaklığa değil.

Buradan çıkan pratik sonuç

Veri hazırlığı iki modelde de **neredeyse aynıdır**: aynı bad tanımı, aynı değişken havuzu, aynı WOE tablosu ikisini de besler. Bu yüzden Random Forest'ı bir **baseline** olarak kurmak ucuzdur — hazırlanan veri hattı aynen XGBoost'a devredilir. Değişen şey modelin ayarı ve validasyon disiplinidir, veri değil.

03 Varyans mı, yanlılık mı?

Test hatası kabaca üç parçadır: **yanlılık² + varyans + indirgenemez gürültü**. Random Forest ikinci terime, XGBoost birinci terime saldırır. Ayarlarının neden zıt yönde olduğunu açıklayan tek cümle budur.



Şekil 3. Temsilî ayrıştırma: bagging varyansı, boosting yanlılığı düşürür.

Boyut	Random Forest	XGBoost
Nasıl iyileşir?	Bağımsız hataların ortalaması sıfıra yaklaşır	Kalan hata her turda biraz daha küçülür
Ağaçların korelasyonu	Düşük olmalı — `max_features` bunun için var	Yüksektir, sorun değil; ardışıklık zaten öyle ister
Gürültülü etikete tepki	Ortalama gürültüyü söndürür — dayanıklı	Gürültüyü de öğrenmeye çalışır — kırılgan
Sinyal zayıfsa	Erken doyar, tavana çarpar	İnce sinyali kovalamaya devam eder
Kapasite artırma yolu	Ağaçları derinleştirmek	Tur eklemek (derinleştirmek değil)

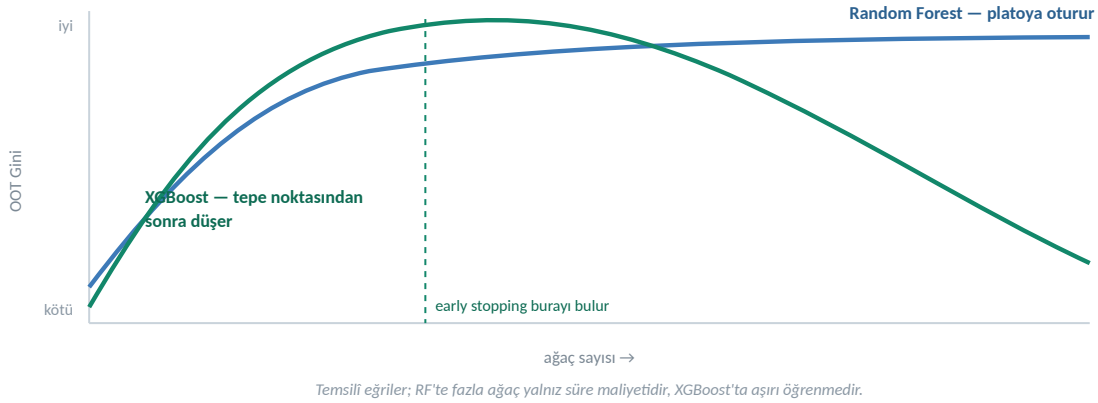
Kredi riskinde bu ne demek?

Başvuru verisinde etiket gürültüsü yüksektir: “bad” tanımı bir eşik kararıdır, aynı profildeki iki müşteri farklı sonuçlanır. Bu ortamda XGBoost'un ince sinyal kovalama yeteneği **hem avantaj hem risktir**. Fark genellikle geliştirme setinde büyük, OOT'de küçük görünür — çünkü kovalanan sinyalin bir kısmı gürültüdür. 9. bölüm bu farkın gerçekte ne kadar olduğunu ölçmenin yolunu verir.

04 Ağaç sayısı ve derinlik

Aynı isimli iki parametre, iki modelde tamamen farklı anlam taşır. En sık yapılan hata, birinden gelen sezgiyi diğerine taşımaktır.

Ağaç sayısı arttıkça OOT performansı ne yapar?



Temsili eğriler; RF'te fazla ağaç yalnız süre maliyetidir, XGBoost'ta aşırı öğrenmedir.

Şekil 4. Aynı parametre adı, iki farklı risk profili.

	Random Forest	XGBoost
`n_estimators`	Bir **yakınsama** parametresidir. Fazlası zarar vermez; yalnız süre ve bellek maliyeti getirir. 300–1000 arası tipiktir.	Bir **kapasite** parametresidir. Fazlası doğrudan aşırı öğrenmedir. Üst sınırı yüksek verilip erken durdurmaya bırakılır.
Erken durdurma	Gerekmez — eğri düşmez. Ağaç sayısını performans yerine süreye göre seçersin.	Zorunludur. Ayrı bir validation setiyle `early_stopping_rounds`.
Derinlik	Derin istenir: `max_depth=None` tipiktir; sınır `min_samples_leaf` ile konur.	Sığ istenir: `max_depth` 3–6. Derin ağaç + çok tur = ezber.
Öğrenme oranı	Yoktur. Her ağacın ağırlığı eşittir (1/M).	Vardır ve en önemli düğmedir. Düşür → tur sayısını yükselt.
Ağaçlar arası bağ	Yok; ağaç ekleyip çıkarabilirsin.	Var; ilk 100 ağacı atmak modeli bozar.

En sık yapılan hata: derinlik sezgisini taşımak

Random Forest'ta çalışan "derin ağaç bırak, ortalama halleder" refleksi XGBoost'ta doğrudan zarar verir: orada ortalama alan bir mekanizma yoktur, ağaçlar **toplanır**. `max_depth=None` gibi bir seçenek zaten yoktur; verilen her ek derinlik kalıcı olarak modele girer.

Ters yönü de aynı derecede yaygındır: XGBoost sezgisiyle Random Forest'a `max_depth=4` vermek. Sığ ve birbirine benzeyen ağaçların ortalaması, ortalamadan beklenen kazancı vermez — model gereksiz yere yetersiz kalır.

05 Aşırı öğrenme ve düzenlileştirme

Random Forest düzenlileştirmeyi **mimarisinden** alır; XGBoost **kayıp fonksiyonuna yazılmış cezalardan**. Biri varsayılan olarak güvenlidir, diğeri güvenli hale getirilir.

Şekil 5. Random Forest'ta fren mimaride, XGBoost'ta parametrededir.

Random Forest gerçekten aşırı öğrenmez mi?

Yaygın ifade "Random Forest aşırı öğrenmez" biçimindedir; doğrusu şudur: **ağaç sayısını artırmak** aşırı öğrenmeye yol açmaz. Model yine de aşırı öğrenebilir — özellikle yapraklar tek gözleme kadar bölünürse ve değişken sayısı gözlem sayısına göre büyükse. Kredi riskinde nadir bad sınıfıyla çalışırken 5 gözlemlik yapraklar üretmek, ortalamanın bile toparlayamayacağı bir gürültü kaynağıdır. Bu yüzden `min_samples_leaf` tek başına en değerli RF parametresidir.

XGBoost'un iki fazla düğmesi

XGBoost'ta hedef fonksiyonun içine yazılmış `reg_lambda` (yaprak skorlarını yumuşatır) ve `gamma` (kazancı eşiği geçmeyen bölünmeyi hiç açmaz) Random Forest'ta karşılığı olmayan iki mekanizmadır. Kredi riskinde `gamma`, gürültülü değişkenlerin ağaca sızmasını engelleyen sessiz bir filtredir; `min_child_weight` ise gözlem sayısı değil **Hessian toplamı** üzerinden çalışır — nadir sınıfta bu, aynı sayının çok daha katı bir kısıt anlamına geldiğini bilmeyi gerektirir.

Tek cümlelik ayrım

Random Forest'ta **yanlış ayar performansı biraz düşürür**; XGBoost'ta yanlış ayar, geliştirme setinde mükemmel görünen ve OOT'de çöken bir model üretir. İkisi arasındaki asıl risk farkı budur — ortalama performans değil, **hata yapıldığında ödenen bedel**.

06 Parametre sözlüğü

Soldaki sütun “ne yapmak istiyorum”, sağdakiler “hangi düğmeyi çevireceğim”. Bazı satırların karşılığı yoktur — asıl bilgi orada saklıdır.

Niyet	Random Forest (sklearn)	XGBoost
Topluluk büyüklüğü	`n_estimators` — büyük = iyi	`n_estimators` + erken durdurma
Ağaç kapasitesini sınırla	`max_depth` (genelde None)	`max_depth` = 3–6
Küçük yaprakları engelle	`min_samples_leaf`, `min_samples_split`	`min_child_weight` (Hessian toplamı)
Kazançsız bölünmeyi engelle	`min_impurity_decrease`	`gamma`
Yaprak skorunu yumuşat (L2)	— (yapraklar ham oran)	`reg_lambda`
L1 ceza	—	`reg_alpha`
Her katkıyı küçült	— (öğrenme oranı yok)	`learning_rate`
Satır örnekle	`bootstrap=True`, `max_samples`	`subsample`
Kolon örnekle	`max_features` — **bölünme başına**	`colsample_bytree` / `bylevel` / `bynode`
Erken durdurma	— (OOB skoru izlenebilir)	`early_stopping_rounds`
Sınıf dengesizliği	`class_weight="balanced_subsample"`	`scale_pos_weight`
Monotonluk kısıtı	`monotonic_cst` (sklearn 1.4+)	`monotone_constraints`
Etkileşim kısıtı	—	`interaction_constraints`
Kategorik değişken	— (ön işleme şart)	`enable_categorical=True`
Paralellik	`n_jobs` — ağaç düzeyinde, neredeyse lineer	`n_jobs` — bölünme düzeyinde

`max_features` ile `colsample_bytree` aynı şey değildir

Random Forest'ta `max_features` **her bölünmede yeniden** çekilir; amacı ağaçları birbirinden ayrıştırmaktır ve modelin temel mekanizmasının parçasıdır. XGBoost'ta `colsample_bytree` ağaç başına bir kez çekilir ve yalnız ek bir düzenlileştiricidir; kapatsan da model çalışır. Sayısal değerleri birbirine kopyalamak bu yüzden anlamsızdır: RF'te `sqrt(p)` tipik bir varsayılanken, XGBoost'ta 0,3 gibi bir oran çoğu zaman fazla agresiftir.

Karşılığı olmayan satırlar en önemlileri

Tabloda üç boş hücre var ve üçü de aynı şeyi söylüyor: **öğrenme oranı, L2 ve erken durdurma** Random Forest'ta yoktur. Bu, RF'i ayarlamayı kolaylaştıran şeydir — ve aynı zamanda tavanını belirleyen şey. XGBoost'un fazladan ayrım gücü tam olarak bu üç düğmeden gelir; bedeli de bu üçünü doğru kurma zorunluluğudur.

07 Başlangıç değerleri

Nadir bad sınıflı, birkaç yüz bin satırlık tipik bir başvuru/portföy veri seti için pratik bir başlangıç noktası. Kesin reçete değil, arama alanının merkezi.

Konu	Random Forest	XGBoost
Topluluk büyüklüğü	`n_estimators = 500` (eğri düzleşene kadar)	`n_estimators = 4000` + erken durdurma
Derinlik	`max_depth = None` (veya 12–20)	`max_depth = 4`
Yaprak alt sınırı	`min_samples_leaf = 200`	`min_child_weight = 10`
Kolon örnekleme	`max_features = "sqrt"`	`colsample_bytree = 0,8`
Satır örnekleme	`bootstrap = True`, `max_samples = 0,8`	`subsample = 0,8`
Öğrenme oranı	—	`learning_rate = 0,03`
L2 / bölünme eşiği	—	`reg_lambda = 5 · gamma = 0,1`
Sınıf dengesizliği	`class_weight = "balanced_subsample"`	`scale_pos_weight = neg/pos`
Erken durdurma	—	100 tur, ayrı validation seti
Metrik	OOB + validation AUC	`eval_metric = "auc"`
Tekrarlanabilirlik	`random_state`, `n_jobs` sabit	`random_state`, `n_jobs`, `tree_method` sabit

Random Forest için iki kritik ayar

`min_samples_leaf` varsayılanı (1) kredi riski için felakettir: nadir bad sınıfında tek gözlemlik yapraklar üretir. Yüz binlerce satırlık portföyde **100–1000 aralığı** daha güvenlidir; bu tek değişiklik hem OOT stabilitesini hem de skor dağılımının düzgünlüğünü belirgin biçimde iyileştirir.

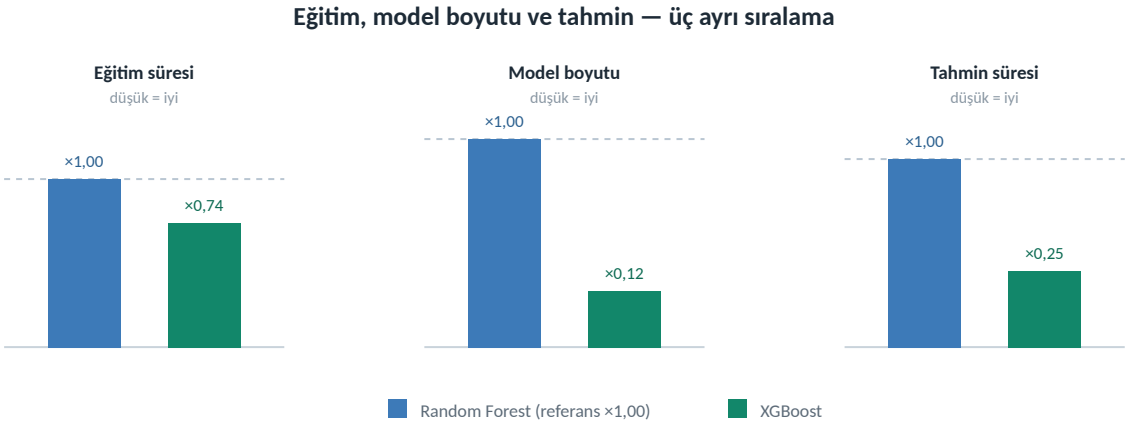
`max_features` sınıflamada varsayılan olarak `"sqrt"`tir; fakat kredi riskinde değişkenlerin çoğu birbiriyle korelasyonlu ve bir kısmı zayıf sinyallidir. Çok sayıda zayıf değişken varsa `"sqrt"`, güçlü değişken sayısı azsa 0,3–0,5 gibi daha yüksek bir oran daha iyi çalışır. Bu, RF'te aramaya değen ikinci parametredir.

XGBoost'ta sıralama önemlidir

Aramayı şu sırayla yap: önce `learning_rate` düşük sabitlenir (0,03), sonra `max_depth` ve `min_child_weight` birlikte aranır, ardından `reg_lambda` ve `gamma`, en son örnekleme oranları. Tur sayısı hiçbir zaman elle aranmaz — onu erken durdurma belirler. Hepsini tek seferde geniş bir gridde aramak, aramanın çoğunu anlamsız köşelerde harcar.

08 Hız, bellek ve ölçeklenme

Eğitim süresi ile tahmin süresi ayrı konulardır ve sıralamaları aynı değildir. Üretimde genellikle ikincisi ile model boyutu daha pahalıya mal olur.



Şekil 6. Temsilî değerler; yön göstermek içindir, benchmark sonucu değildir.

Eğitim: paralellik RF'in, verimlilik XGBoost'un lehine

Random Forest ağaç düzeyinde **utanç verici derecede paralel**dir; 16 çekirdekte 16 ağaç aynı anda büyür. Ama her ağaç tam veriye ve tam derinliğe kadar iner. XGBoost ağaçları sırayla büyütür — fakat histogram tabanlı bölünme arama, sığ ağaçlar ve erken durdurma sayesinde toplam iş çok daha azdır. Pratikte tipik bir kredi riski veri setinde ikisi aynı mertebede kalır; RF çekirdek sayısı arttıkça, XGBoost veri büyüdükçe öne geçer.

Model boyutu ve tahmin: fark burada büyür

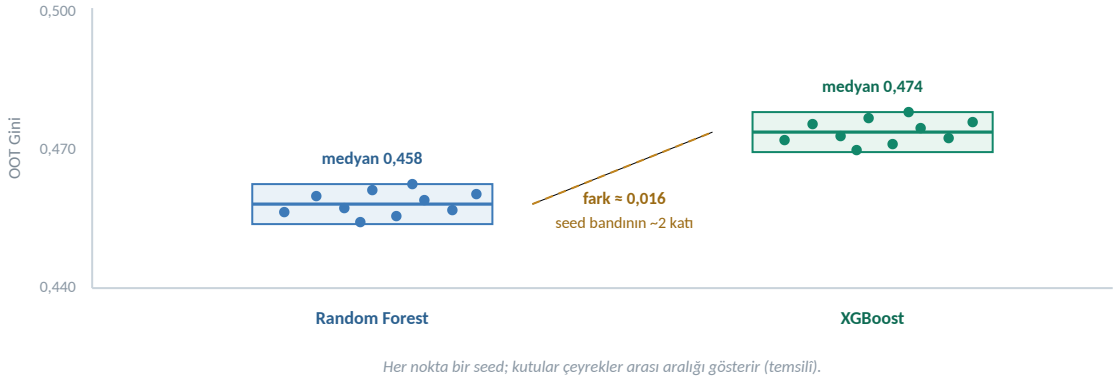
500 adet tam büyümüş ağaç, yüz binlerce düğüm demektir; serileştirilmiş model yüzlerce megabayta çıkabilir. 400 adet derinlik-4 XGBoost ağacı ise en fazla 6.400 yapraktır. Gerçek zamanlı karar veren bir başvuru motorunda bu fark hem gecikmede hem de bellek ayak izinde doğrudan hissedilir. RF'in tahmin maliyetini düşürmenin tek yolu ağaç sayısını veya derinliği kısmaktır — ikisi de performanstan götürür.

GPU tarafı

XGBoost'ta `device="cuda"` olgun bir yoldur; scikit-learn Random Forest'ta GPU desteği yoktur (cuML gibi ayrı kütüphaneler gerekir). Ancak birkaç yüz bin satırlık tipik bir kredi riski veri setinde GPU çoğu zaman kazandırmaz; veri aktarım maliyeti avantajı yer.

09 Doğruluk farkı gerçekte ne kadar?

Diğer karşılaştırmadan farklı olarak burada **sistematik bir yön** vardır: iyi ayarlanmış boosting tabular veride genellikle kazanır. Kritik soru, farkın büyüklüğü ve gürültüye oranıdır.



Şekil 7. Temsilî dağılım: fark gürültü bandını aşıyorsa bulgudur.

Aynı modeli yalnız `random_state` değiştirerek 20 kez eğitmek, OOT Gini'de tipik olarak **±0,004–0,010** bandında bir dalgalanma üretir. Literatürdeki tabular karşılaştırmalarda iyi ayarlanmış boosting ile Random Forest arasındaki tipik fark **0,01–0,03 Gini** mertebesindedir — yani genellikle bu bandın üzerindedir, ama uçurum da değildir.

Farkın kapandığı durumlar

· Değişkenler zaten dikkatle WOE'ye çevrilmişse — ön işleme, modelin yapabileceği işin bir kısmını üstlenir.
· Veri küçükse (< 50 bin satır) veya bad oranı çok düşükse — XGBoost'un ince sinyali gürültüden ayırt etme şansı azalır.
· Etiket gürültüsü yüksekse — RF'in ortalaması bu durumda avantaja dönüşür.
· XGBoost'a eşit arama bütçesi verilmemişse — o zaman ölçülen şey model farkı değil, ayar farkıdır.

Karar kuralı

Birini diğerine tercih etmeden önce: her ikisini de **en az 5 farklı seed** ve **en az 2 farklı OOT penceresi** ile çalıştır. Farkın medyanı seed dağılımının genişliğinin yaklaşık iki katından küçükse, seçimini performansa değil; **bakım yükü, açıklanabilirlik, skorlama maliyeti ve ekip aşinalığı** gibi ölçütlere dayandır.

10 Kategorik değişken ve eksik değer

Kredi riski verisinin karakterini kategorik alanlar belirler. İki modelin pratikteki en büyük ayrımı burada ortaya çıkar — ve bu ayrım modelin zekâsıyla değil, kütüphane desteğiyle ilgilidir.

Konu	Random Forest (sklearn)	XGBoost
Native kategorik	Yok — sayısala çevirmek zorunlu	Var — `enable_categorical=True` + pandas `category`
Yüksek kardinalite	One-hot ile ciddi zorlanır	Bölüm (partition) tabanlı bölme ile idare eder
Eksik sayısal	sklearn 1.4+ destekler; öncesinde impute şart	Her bölünmede varsayılan yön öğrenilir
Eksik kategorik	Ayrı kategori olarak kodlanmalı	Ayrı kategori olarak işlenir
Ölçekleme gerekir mi?	Hayır	Hayır
Aykırı değer	Sıralamaya duyarlı — etkisi sınırlı	Aynı

One-hot, Random Forest'ta neden özellikle zararlıdır?

Şube kodunu (700+ seviye) one-hot yaptığında 700 seyrek kolon üretirsin. Random Forest her bölünmede yalnız `sqrt(p)` kadar aday değişken çeker; bu 700 kolon aday havuzunu şişirir ve **gerçekten güçlü değişkenlerin seçilme olasılığını düşürür**. Üstelik tek bir seyrek kolonun tek başına kazancı düşüktür, bu yüzden ağaçlar onları çoğu zaman hiç kullanmaz — bilgi kaybolur.

Doğrusu: yüksek kardinaliteli alanları **WOE'ye çevir** veya iş anlamına göre grupla (şube → bölge/segment, meslek → meslek grubu). Bu tek adım, Random Forest ile XGBoost arasındaki farkın önemli bir kısmını kapatır.

WOE kullanıyorsan sızıntıya dikkat

WOE bir hedef istatistiğidir. Tüm veri üzerinden bir kez hesaplanan WOE tablosu, kredi riski projelerinde **en sık görülen sessiz sızıntı kaynağıdır** ve bu risk her iki modelde de aynıdır. İstatistiği yalnız train diliminde hesapla, fold içinde yeniden üret ve düşük frekanslı kategorileri düzleştir (smoothing).

Kredi riskinde eksiklik bir bilgidir

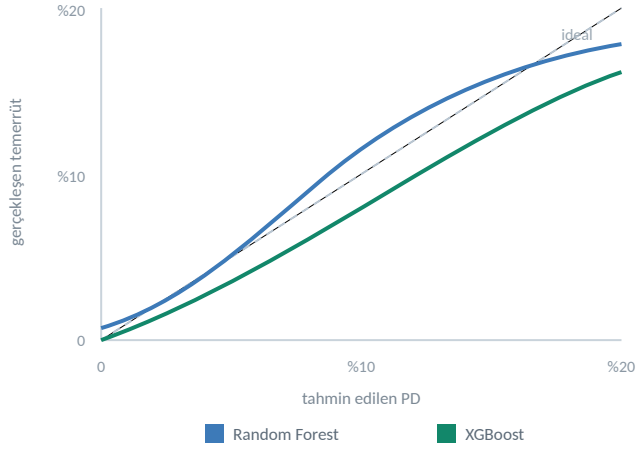
"KKB skoru yok" bir bilinmeyen değil, çoğu zaman **yeni müşteri** demektir; "gelir beyanı yok" bir başvuru kanalını işaret eder. Otomatik yönetimin yanına açık bir `*_IS_MISSING` bayrağı eklemek genellikle hem performansı hem açıklanabilirliği artırır. Random Forest'ta bu ayrıca **zorunluluğa yakındır**: eski sürümlerde impute ettiğin değer (medyan gibi) eksikliğin taşıdığı sinyali tamamen siler.

Bir kaynak sistem devre dışı kaldığında eksiklik oranı bir gecede yükselir. Model eksikliği "düşük risk" olarak öğrenmişse skor sessizce kayar. Eksik oranlarını CSI ile birlikte izle; bu, iki modelde de aynı derecede geçerli bir üretim riskidir.

11 Kredi riskinde kalibrasyon

İkisi de olasılık üretir; hiçbiri kalibre edilmiş PD üretmez. Fakat bozulmanın **şekli** birbirinden farklıdır ve bu, kalibrasyon katmanının seçimini etkiler.

İki farklı sapma profili



Okuma

Random Forest oy oranı üretir; uçlara nadiren gider. Skor dağılımı ortaya toplanır, üst dilimde ayrışma zayıflar.

XGBoost logit ölçeğinde çalışır; uçları rahat üretir. scale_pos_weight verilirse riski sistematik olarak abartır.

İkisi de kalibrasyonsuz PD olarak kullanılamaz.

Şekil 8. Temsilî gösterim; gerçek sapma veri ve ayarlara göre değişir.

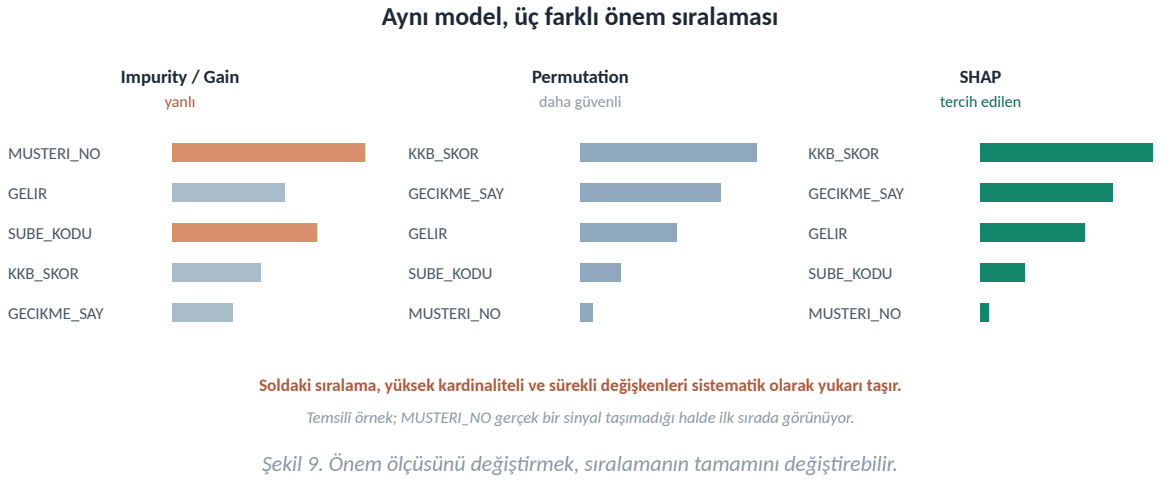
	Random Forest	XGBoost
Çıktının doğası	Ağaçların oy oranı — [0,1] aralığında sıkışık	Logit toplamının sigmoidi — uçlara ulaşabilir
Tipik sapma	Uçlarda sıkışma; üst dilimde çözünürlük kaybı	Ayara duyarlı; ağırlık verilirse sistematik abartma
Kritik parametre	`min_samples_leaf`, `n_estimators`	`reg_lambda`, `max_depth`, `scale_pos_weight`
Uygun kalibrasyon	Isotonic (monotonik ama esnek sapma)	Platt/lojistik çoğu zaman yeterli
Skor granülerliği	Sınırlı — 500 ağaçta 501 olası değer	Sürekliye yakın

Değişmeyen kural

Hangi modeli seçersen seç, `scale_pos_weight` veya `class_weight` kullandıysan **çıktı PD değildir**. Ayrı bir dilimde kalibrasyon katmanı öğren (lojistik / isotonic), OOT'de doğrula ve dilim bazında gözlenen–beklenen testini yap. Model seçimi bu adımı ortadan kaldırmaz.

12 Değişken önemi: en sık düşülen tuzak

Her iki kütüphanenin de kutudan çıkan değişken önemi ölçüsü **yanlıdır**. Kredi riskinde bu yanlılık yalnız akademik bir kusur değil; değişken eleme kararlarını doğrudan bozar.



Şekil 9. Önem ölçüsünü değiştirmek, sıralamanın tamamını değiştirebilir.

Yanlılık nereden geliyor?

Impurity tabanlı önem (RF'in `feature_importances_`'ı) ve XGBoost'un `gain` metriği, aynı kusuru paylaşır: **bölünme fırsatı çok olan değişken daha çok kazanç biriktirir**. Çok seviyeli kategorik ve sürekli değişkenler, tesadüfen iyi bir eşik bulma şansına ikili değişkenlerden daha fazla sahiptir. Random Forest'ta etki daha büyüktür çünkü ağaçlar derindir ve her bölünme toplama katkı yapar.

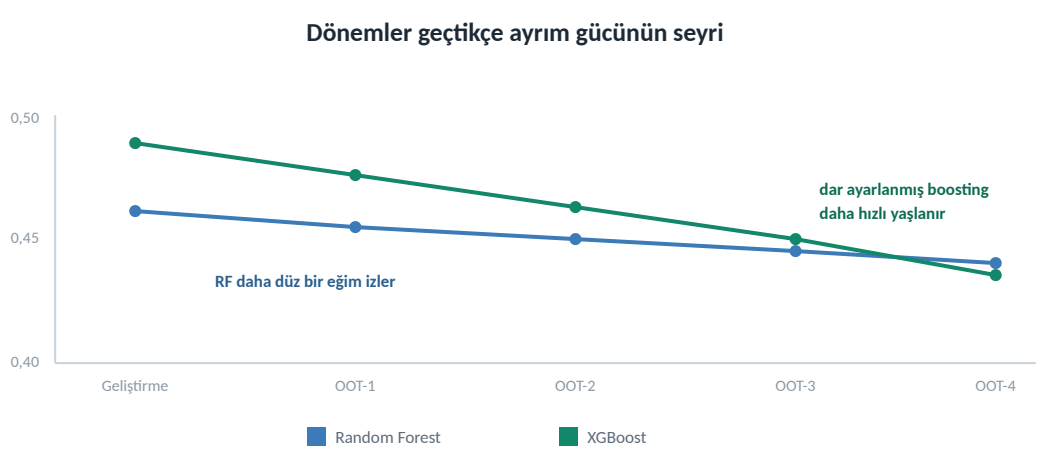
Ne kullanmalı?

· **SHAP** — ikisinde de TreeSHAP ile hızlı ve tutarlıdır; hem sıralama hem yön verir. Varsayılan tercih budur.
· **Permutation importance** — model-agnostiktir ve OOT diliminde hesaplandığında gerçekten anlamlıdır. Korelasyonlu değişken gruplarında dikkatli ol: iki değişken de birbirini telafi ederse ikisi de önemsiz görünür.
· **Impurity / gain** — yalnız hızlı bir ön bakış için. Değişken eleme kararını buna dayandırma.

> **Kredi riskinde pratik sonuç**
>
> Random Forest'ı "ucuz değişken eleme aracı" olarak kullanmak yaygın bir alışkanlıktır ve fena bir fikir değildir — fakat **varsayılan importance ile yapılırsa** şube kodu, müşteri numarası, tarih türevleri gibi yüksek kardinaliteli alanları yukarı taşır, gerçek sinyalleri aşağı iter. Aynı işi permutation veya SHAP ile yaptığında liste belirgin biçimde değişir.

13 Stabilite ve model riski

Kredi riskinde model bir kez seçilip bitmez; iki yıl boyunca izlenir, savunulur ve denetlenir. Seçim kriterine bu ufku da katmak gerekir.



Şekil 10. Temsili seyir: yüksek başlangıç, daha dik yaşlanma anlamına gelebilir.

Boyut	Random Forest	XGBoost
Monotonluk kısıtı	Var (sklearn 1.4+), görece yeni	Var, olgun
Etkileşim kısıtı	Yok	Var
SHAP desteği	TreeSHAP — ağaçlar derin, hesap ağır	TreeSHAP — hızlı
Skor stabilitesi	Yüksek — ortalama ani sıçrama üretmez	Ayara duyarlı
Tekrarlanabilirlik	Seed + `n_jobs` sabitlenmeli	Seed + `n_jobs` + `tree_method` sabitlenmeli
Denetime anlatma	"Ağaçların oyu" — kolay	"Ardışık düzeltmelerin toplamı" — daha zor
Ekip aşinalığı	Çok yaygın	Çok yaygın

OOB skoru OOT değildir

Random Forest'ın en sevilen özelliklerinden biri, bedava gelen **out-of-bag** tahminidir. Kredi riskinde bu bir tuzaktır: OOB, aynı zaman diliminden rastgele ayrılmış gözlemleri kullanır. Portföy kaymasını, kampanya etkisini ve makro değişimi **ölçmez**. OOB'yi hızlı bir sağlık kontrolü olarak kullan; kararı her zaman zaman bazlı OOT penceresiyle ver.

Denetim tarafının sorusu

Bağımsız validasyon biriminin ilk soracağı şey model adı değil, **"aynı sonucu yeniden üretebiliyor musunuz?"** olacaktır. İkisinde de seed'i, sürümü, veri kesitini ve parametreleri kilitle. İş parçacığı sayısı bile sonucu az miktarda değiştirebilir; kritik modelde bunu da sabitle ve test et.

14 Python: aynı problem, iki model

Aynı veri, aynı bölme, aynı metrik. Değişen yalnız modelin ayar felsefesi.

```
# ---------- ORTAK KURULUM ----------
neg, pos = (y_tr == 0).sum(), (y_tr == 1).sum()
spw = neg / max(pos, 1)
cat_cols = ["SUBE", "MESLEK", "IL", "URUN", "KANAL"]

# ---------- RANDOM FOREST ----------
# sklearn native kategorik desteklemez: WOE veya gruplanmış kod ver.
from sklearn.ensemble import RandomForestClassifier

X_tr_rf = woe_transform(X_tr, cat_cols, fit=True)   # yalnız train'de fit!
X_va_rf = woe_transform(X_va, cat_cols, fit=False)

rf = RandomForestClassifier(
    n_estimators=500,               # eğri düzleşene kadar; fazlası zarar vermez
    max_depth=None,                # derin ağaç istenir — ortalama frenler
    min_samples_leaf=200,           # varsayılan 1 kredi riski için felaket
    max_features="sqrt",           # bölünme BAŞINA aday sayısı
    bootstrap=True, max_samples=0.8,
    class_weight="balanced_subsample",
    oob_score=True,                # bedava sağlık kontrolü (OOT değil!)
    random_state=42, n_jobs=8
)
rf.fit(X_tr_rf, y_tr)
# erken durdurma YOK — ağaç sayısı süreye göre seçilir

# ---------- XGBoost ----------
from xgboost import XGBClassifier

X_tr_x = X_tr.copy(); X_va_x = X_va.copy()
for c in cat_cols:                     # native kategorik için category dtype şart
    X_tr_x[c] = X_tr_x[c].astype("category")
    X_va_x[c] = X_va_x[c].astype(X_tr_x[c].dtype)

xgb = XGBClassifier(
    objective="binary:logistic", eval_metric="auc", tree_method="hist",
    enable_categorical=True, max_cat_to_onehot=8,
    n_estimators=4000, learning_rate=0.03, max_depth=4,
    min_child_weight=10, gamma=0.1, reg_lambda=5.0,
    subsample=0.8, colsample_bytree=0.8, scale_pos_weight=spw,
    early_stopping_rounds=100,      # RF'te karşılığı olmayan zorunluluk
    random_state=42, n_jobs=8
)
xgb.fit(X_tr_x, y_tr, eval_set=[(X_va_x, y_va)], verbose=False)
```

Koddaki üç bilinçli tercih

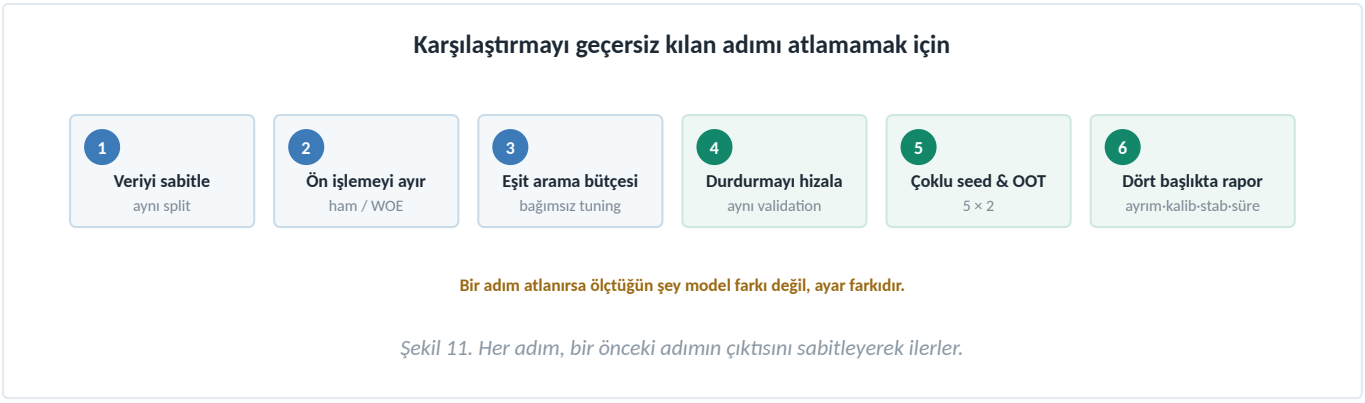
RF'e WOE, XGBoost'a ham kategorik verildi. Bu bir haksızlık değil, her modelin kendi en iyi haliyle çalıştırılmasıdır. Adil karşılaştırma için ikisini de **aynı iki kolda** (ham / WOE) denemek gerekir — 15. bölüm.

RF'te `max_depth=None`, XGBoost'ta 4. Bu sayıları eşitlemek en yaygın hatadır.

RF'te `oob_score` açık ama karar için kullanılmıyor. OOB hızlı bir sağlık göstergesidir; model kararı zaman bazlı OOT ile verilir.

15 Adil karşılaştırma protokolü

Model karşılaştırmalarının çoğu, karşılaştırma değil **ayar farkı ölçümüdür**. Aşağıdaki sıra bunu engeller.



Şekil 11. Her adım, bir önceki adımın çıktısını sabitleyerek ilerler.

1. Veriyi sabitle. Aynı train / validation / OOT bölmesi, aynı müşteri bazlı split, aynı bad tanımı. Bölme farkı, model farkından büyüktür.

2. Ön işlemeyi ayır. Kategorikleri iki kolda dene: (a) ham — yalnız XGBoost native destekler; (b) WOE ile kodlanmış — ikisi de aynı girdiyi görür. Karar için asıl anlamlı kol ikincisidir.

3. Her modele eşit arama bütçesi ver. RF'in 3 parametresi var diye ona daha az deneme vermek adil görünür ama değildir; aynı sayıda deneme, aynı arama yöntemi.

4. Durdurmayı hizala. XGBoost erken durdurmayı validation setinde yapar; RF'te ağaç sayısı eğri düzleşene kadar artırılır. **OOT hiçbirinde durdurma için kullanılmaz.**

5. Çoklu seed ve çoklu OOT ile tekrarla. En az 5 seed, en az 2 dönem.

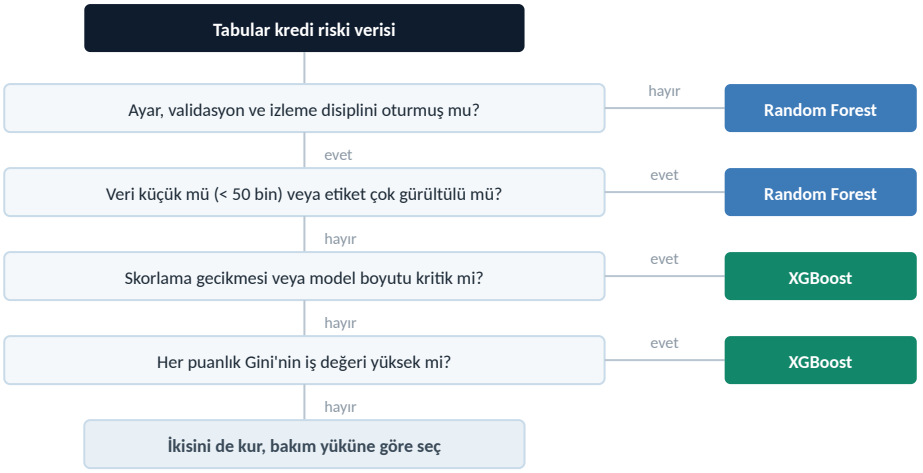
6. Dört başlıkta raporla: ayrım, kalibrasyon, stabilite, süre. Tek metrikli karşılaştırma karar için yetersizdir.

```
import numpy as np, pandas as pd
from sklearn.metrics import roc_auc_score, average_precision_score, brier_score_loss

def degerlendir(model, X, y, kind):
    p = model.predict_proba(X)[:, 1]
    auc = roc_auc_score(y, p)
    return {
        "model": kind,
        "gini": 2 * auc - 1,
        "pr_auc": average_precision_score(y, p),
        "brier": brier_score_loss(y, p),          # kalibrasyonu da ölç
        "top1_capture": y[p >= np.quantile(p, 0.99)].mean() / y.mean()
    }

sonuc = []
for seed in [42, 7, 2024, 101, 555]:          # seed varyansını ölç
    for oot in oot_pencereleri:                # en az iki farklı dönem
        for ad, kur in kurucular.items():        # {"rf": fn, "xgb": fn}
            m = kur(seed)                        # her biri KENDİ aramasıyla ayarlanmış
            m.fit(*egitim_verisi)
            r = degerlendir(m, *oot, ad)
            r["seed"], r["oot"] = seed, oot.ad
            sonuc.append(r)

df = pd.DataFrame(sonuc)
ozet = df.groupby("model")["gini"].agg(["median", "std", "min", "max"])
# Karar kuralı: medyan farkı, seed std'sinin ~2 katından küçükse "fark yok" say.
```

16 Hangisini ne zaman seçmeli?

Performans yakınsa karar, verinin ve ortamın özelliklerine göre verilir.

Veriden ve kısıtlardan başlayan seçim akışı



Akış bir kural değil, başlangıç noktasıdır; nihai karar 15. bölümdeki protokolle verilir.

Şekil 12. Seçim, performanstan önce ortamla ilgilidir.

Durum	İlk deneme	Neden
Hızlı baseline, birkaç günde sonuç	Random Forest	Varsayılanlar güvenli; ayar aramadan makul bir taban verir.
Şampiyon model, ayrım gücü kritik	XGBoost	Tavanı daha yüksek; disiplin varsa farkı toplayabilirsin.
Küçük veri (< 50 bin) veya çok düşük bad oranı	Random Forest	Boosting bu rejimde gürültüyü kolayca ezberler.
Gerçek zamanlı başvuru motoru	XGBoost	Sığ ağaçlar; model küçük, skorlama ucuz.
Değişken eleme / keşif aşaması	Random Forest	Hızlı, paralel ve ayar hatasına dayanıklı — ama SHAP ile oku.
Model riski onayı ağır, kısıt gerekiyor	XGBoost	Monotonluk ve etkileşim kısıtları daha olgun.
Değişkenler zaten WOE'ye çevrilmiş skorkart hattı	Fark küçülür	Ön işleme işin bir kısmını üstlenir; süre ve aşinalık belirler.

Pratik öneri

Random Forest'ı **şampiyon değil, referans** olarak kur. Her yeni modelde ilk gün bir RF eğit ve Gini'sini not et: XGBoost'un ürettiği fark bu referansın üzerinde anlamlı biçimde durmuyorsa, ya ayar eksiktir ya da veride o kadar sinyal yoktur. Bu tek alışkanlık, hem aşırı ayarlanmış modelleri hem de boşa harcanan haftaları erken yakalar.

17 Yaygın tuzaklar

Karşılaştırmayı ya da modeli sessizce bozan hatalar.

Tuzak	Ne olur?	Doğrusu
RF'te `min_samples_leaf`'i varsayılanda bırakmak	Nadir bad sınıfında tek gözlemlik yapraklar; skor dağılımı dişli çıkar.	Büyük portföyde 100–1000 aralığını dene.
XGBoost'a RF derinlik sezgisini taşımak	Derin ağaç + çok tur = geliştirmede mükemmel, OOT'de çöküş.	`max_depth` 3–6, erken durdurma zorunlu.
RF'e erken durdurma aramak	Boşa harcanan zaman; eğri zaten düşmez.	Ağaç sayısını eğri düzleşene kadar artır, sonra süreye bak.
Aynı parametreleri ikisine kopyalamak	Ölçekler farklı olduğu için ayar değil, şans ölçülür.	Her modele eşit bütçeyle bağımsız arama.
OOB'yi OOT sanmak	Zaman kaymasını görmezsin; performans iyimser çıkar.	OOB sağlık kontrolü, karar zaman bazlı OOT ile.
Varsayılan importance ile değişken elemek	Yüksek kardinaliteli alanlar yukarı, gerçek sinyal aşağı iner.	SHAP veya OOT'de permutation importance.
Yüksek kardinaliteyi RF'e one-hot vermek	Aday havuzu şişer, güçlü değişkenler seçilemez.	WOE uygula veya iş anlamına göre grupla.
WOE'yi tüm veride hesaplamak	Sessiz hedef sızıntısı; validasyon iyimser çıkar.	Yalnız train diliminde, fold içinde yeniden üret.
OOT'yi erken durdurmada kullanmak	İkisinde de aynı hata; performans iyimser.	Ayrı validation; OOT yalnız bir kez ölçülür.
Çıktıyı PD sanmak	Sınıf ağırlığı verildiyse mutlak seviye anlamsızdır.	Ayrı dilimde kalibrasyon katmanı öğren ve doğrula.
Tek seed ile karar vermek	Gürültü, bulgu sanılır.	Çoklu seed ve çoklu OOT ile medyan ± std raporla.
Sürümü kilitlememek	Varsayılan değişikliği modeli sessizce oynatır.	Sürümü ve seed'i model dokümanına yaz, ortamı sabitle.

Hepsinin ortak paydası

Listedeki hataların hiçbiri model hatası değildir. İkisi de kendilerinden isteneni yapar; sorun **ne istediğimizi yanlış ifade etmemizden** çıkar.

Hızlı referans kartı

Tek sayfada iki model.

	Random Forest	XGBoost
Topluluk mantığı	Bagging — paralel, bağımsız	Boosting — ardışık, bağımlı
Hedeflediği hata	Varyans	Yanlılık
Ağaç şekli	Derin, tam büyümüş	Sığ (derinlik 3–6)
Ağaç sayısı	`n_estimators` = 300–1000 (fazlası zararsız)	`n_estimators` = üst sınır + erken durdurma
Yaprak alt sınırı	`min_samples_leaf` = 100–1000	`min_child_weight` = 5–20
Kolon örnekleme	`max_features` = "sqrt" (bölünme başına)	`colsample_bytree` = 0,7–0,9
Öğrenme oranı	Yok	`learning_rate` = 0,02–0,05
L2 / bölünme cezası	Yok	`reg_lambda`, `gamma`
Kategorik	Ön işleme şart (WOE / gruplama)	`enable_categorical`
Sınıf dengesizliği	`class_weight`	`scale_pos_weight`
Eğitim hızı	Paralellikte güçlü	Toplam işte verimli
Tahmin hızı / boyut	Yavaş, model büyük	Hızlı, model küçük
Ezberleme riski	Düşük	Yüksek — ayara bağlı
Varsayılan güvenliği	Yüksek	Düşük
Ayar yükü	Düşük — 2–3 düğme	Yüksek — 8–10 düğme
İlk tercih olduğu yer	Baseline, keşif, küçük/gürültülü veri	Şampiyon model, düşük gecikme, büyük veri

Tek cümlelik özet

Random Forest **güvenliği ve düşük bakım maliyetini**, XGBoost **aynı veriden çıkarılabilecek son birkaç Gini puanını** satın aldığın yerdir. Kredi riski hattında ikisi de kabul edilebilir bir modeldir; farkı yaratan model değil, **validasyon tasarımı, kalibrasyon ve izleme disiplinidir**.

18 Kaynaklar

Resmî dokümantasyon ve yöntemleri tanıtan özgün makaleler.

1 **scikit-learn — Ensemble methods, RandomForestClassifier**
scikit-learn.org/stable/modules/ensemble.html

2 **XGBoost Documentation — Parameters, Tree Methods, Categorical Data**
xgboost.readthedocs.io/en/stable/

3 **Breiman — Random Forests (Machine Learning, 2001)**
Bagging + rastgele değişken alt kümesinin tanıtıldığı makale

4 **Chen & Guestrin — XGBoost: A Scalable Tree Boosting System (KDD 2016)**
arxiv.org/abs/1603.02754

5 **Strobl et al. — Bias in random forest variable importance measures (BMC Bioinformatics, 2007)**
Impurity tabanlı önem yanlılığının kaynağı

6 **Niculescu-Mizil & Caruana — Predicting Good Probabilities with Supervised Learning (ICML 2005)**
Ağaç topluluklarının kalibrasyon davranışı

7 **Grinsztajn et al. — Why do tree-based models still outperform deep learning on tabular data? (NeurIPS 2022)**
Tabular veride ağaç toplulukların konumu

8 **BCDataWorks Çalışma Notları № 03 — XGBoost · LightGBM · CatBoost**
Boosting kütüphaneleri arasındaki tasarım farkları

Kapsam notu

Bu doküman **tabular ikili sınıflandırma** ve kredi riski bağlamına odaklanır. Sıralama (ranking), çok sınıflı, sağkalım, zaman serisi ve dağıtık eğitim senaryolarında iki modelin göreli konumu değişebilir. Random Forest tarafında anlatım scikit-learn uygulamasını esas alır; ranger veya H2O gibi uygulamalar kategorik değişkenleri native olarak işleyebilir ve buradaki bazı kısıtlar onlarda geçerli değildir.

Son söz

Model seçimi, bir kredi riski modelinin başarısını belirleyen ilk on faktörden biri bile değildir. Hedef tanımı, validasyon tasarımı, değişken kalitesi, kalibrasyon ve izleme; ikisi arasındaki farkın kat kat üzerinde etki taşır. Bu dokümanın amacı seçimi hızlıca ve gerekçeli biçimde kapatıp asıl işe dönmektir.

BCDataWorks Çalışma Notları № 04 · Ağustos 2026

— ÇALIŞMA SONU —