GELİŞMİŞ AĞAÇ YÖNTEMLERİ

Bagging ve Boosting Kavramları:

Bagging (Bootstrap Aggregating)

Bagging, birden fazla modelin **paralel** olarak eğitildiği bir tekniktir. Temel fikir, **rastgelelik** ekleyerek modellerin birbirinden bağımsız hatalar yapmasını sağlamak ve bu hataları birleştirerek genel hatayı azaltmaktır.

- **Nasıl Çalışır?**
    1. Orijinal veri setinden **yedeklemeli rastgele örnekleme (bootstrap sampling)** ile farklı alt kümeler oluşturulur. Yani, aynı veri noktası birden fazla alt kümede yer alabilir.
    2. Her bir alt küme üzerinde **bağımsız olarak** ayrı bir temel model (örn. karar ağacı) eğitilir.
    3. Sonuç olarak, her model kendi tahmini yapar. Sınıflandırma problemlerinde **çoğunluk oylaması**, regresyon problemlerinde ise **ortalama alma** ile nihai tahmin elde edilir.
- **Ne Zaman Kullanılır?** Genellikle **yüksek varyansa sahip (overfitting'e yatkın) modellerin** performansını iyileştirmek için etkilidir. Varyansı azaltırken sapmayı (bias) artırmaz.
- **Anahtar Özellik:** Paralel eğitim, varyans azaltma.
- **Örnek Algoritma: Random Forest (Rastgele Orman)**, Bagging'in en bilinen örneğidir. Random Forest'ta, sadece veri alt kümeleri değil, aynı zamanda her bir düğümde kullanılan özellikler de rastgele seçilir, bu da varyansın daha da azalmasına yardımcı olur.

Veri Bilimi Caselerinden Örnek: Müşteri Terkini Tahmin Etme

Bir e-ticaret şirketinde müşteri terkini (churn) tahmin etmek istediğinizi düşünün. Müşteri verileri karmaşık olabilir ve tek bir karar ağacı modeli kolayca aşırı uyum sağlayabilir (overfit olabilir).

- **Bagging Uygulaması:** Müşteri verilerinizden (yaş, cinsiyet, satın alma geçmişi, sayfa ziyaretleri vb.) **birden fazla bootstrap örneği** alırsınız. Her bir örnek üzerinde ayrı bir **karar ağacı** eğitirsiniz. Diyelim ki 100 tane

karar ağacı oluşturdunuz. Yeni bir müşterinin terkedip terketmeyeceğini tahmin etmek istediğinizde, bu 100 ağacın her birinden tahmin alırsınız. Eğer 100 ağacın 70'i müşterinin terk edeceğini tahmin ediyorsa, Bagging modeli de müşterinin terk edeceğini öngörür. Bu yaklaşım, tek bir ağacın yapabileceği hataları **geniş bir konsensüsle** düzeltir.

Boosting

Boosting, birden fazla modelin **sıralı (sequential)** olarak eğitildiği bir tekniktir. Temel fikir, her yeni modelin, bir önceki modelin **yanlış tahmin ettiği veri noktalarına daha fazla odaklanarak** eğitilmesidir. Yani, her model bir önceki modelin hatalarını düzeltmeye çalışır.

- **Nasıl Çalışır?**
    1. İlk model tüm veri seti üzerinde eğitilir.
    2. Sonraki modeller, bir önceki modelin **yanlış sınıflandırdığı veya büyük hata yaptığı** veri noktalarına daha fazla "ağırlık" vererek eğitilir.
    3. Bu süreç, belirli bir sayıda model eğitilene veya performans doygunluğa ulaşana kadar tekrarlanır.
    4. Nihai tahmin, tüm modellerin ağırlıklı ortalaması alınarak elde edilir. Başarılı modellerin tahminleri daha fazla ağırlık taşır.
- **Ne Zaman Kullanılır?** Genellikle **yüksek sapmaya sahip (underfitting'e yatkın) modellerin** performansını artırmak için etkilidir. Sapmayı (bias) azaltırken varyansı hafifçe artırabilir.
- **Anahtar Özellik:** Sıralı eğitim, hatalara odaklanma, sapma azaltma.
- **Örnek Algoritmalar: AdaBoost, Gradient Boosting (GBM), XGBoost, LightGBM, CatBoost.**

Veri Bilimi Caselerinden Örnek: Konut Fiyat Tahmini

Bir emlak şirketinin, evlerin özelliklerine (metrekare, oda sayısı, konum vb.) göre satış fiyatlarını tahmin etmek istediğini düşünün.

- **Boosting Uygulaması:**
    1. Öncelikle, ev özelliklerini kullanarak bir **ilk model** (örn. zayıf bir karar ağacı) eğitirsiniz ve tüm evlerin fiyatlarını tahmin edersiniz.

2. Modelin **büyük hatalar yaptığı** (fiyatı çok yanlış tahmin ettiği) evlere **daha fazla ağırlık** atarsınız. Örneğin, 500.000 dolarlık bir evi 200.000 dolar tahmin eden model, o eve daha fazla "odaklanmalıdır".
3. İkinci bir model eğitilirken, ilk modelin hatalı olduğu bu evlere daha fazla dikkat ederek eğitilir.
4. Bu süreç, her yeni modelin bir önceki modelin düzeltemediği hataları düzeltmeye çalıştığı **sıralı bir şekilde** devam eder.
5. Sonunda, tüm bu ardışık modellerin tahminleri birleştirilerek (ağırlıklandırılarak) **çok daha doğru bir konut fiyatı tahmini** elde edilir. Bu, modelin özellikle zor vakalarda daha iyi öğrenmesini sağlar.

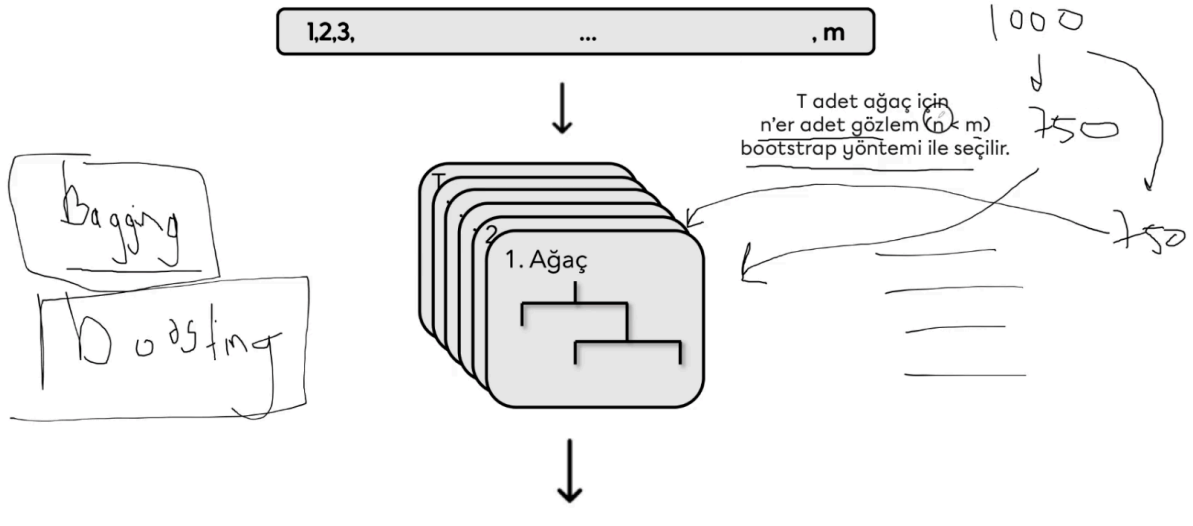
Rastgele Ormanlar (Random Forests)

- **Niye Random:**
  - When constructing a Random Forest, **we don't train each decision tree on the entire original dataset**. Instead, for each tree, a **bootstrap sample (a random sample with replacement) of the training data is drawn.**
  - This "**sampling with replacement**" introduces diversity among the training sets for different trees, making each tree unique. If all trees saw the same data, they would likely learn very similar patterns and errors, defeating the purpose of ensemble learning.
  - This is arguably the more critical source of "randomness" and what truly differentiates Random Forests from simple bagging of decision trees.
- **Why are these randomizations important?:**
  - **Reduces Variance and Prevents Overfitting:** Individual decision trees are prone to overfitting because they can easily learn the noise in the training data. By introducing randomness in both data sampling and feature selection, we ensure that the individual trees in the forest are diverse and less correlated with each other.
  - **Increases Robustness:** The random sampling of data points makes the model more robust to outliers and noisy data.

- **Handles High-Dimensional Data:** Random feature selection is particularly beneficial in datasets with many features, as it avoids any single dominant feature from dictating all the splits across all trees. This helps explore more diverse patterns in the data.
- **Improved Accuracy:** The combined effect of reduced variance and increased diversity typically leads to a more accurate and stable predictive model compared to a single decision tree.

- Değişken ve gözlem seçiminde rassallık var
- Bagging -Bootstrap Aggregation (Breiman, 1996) ile Random Subspace (Ho, 1998) yöntemlerinin birleşimi ile oluşmuştur.
  - CART aşırı öğrenmeye yönelimi var.
- Ağaçlar için gözlemler bootstrap rastgele örnek seçim yöntemi ile değiskenler random subspace yöntemi ile seçilir.
- Karar ağacının her bir düğümünde en iyi dallara ayırıcı (bilgi kazancı) değisken tüm değişkenler arasından rastgele seçilen daha az sayidaki degisken arasindan seçilir.
- Ağaç oluşturmada ver setinin 2/3' kullanılır. Dışarıda kalan veri ağaçlarin performans degerlendirmesi ve değişken öneminin belirlenmesi için kullanılır.
- Her düğüm noktasında rastgele değişken seçimi yapılır. (Regresyon'da p/3, siniflama'da karekök p)

Bagging Metodu:



T adet karar ağacı modelinin ürettiği T adet tahmin değerini bir araya getir.

Bagging vs Boosting: (Çok Önemli Mülakat sorusu)

Feature	Bagging (e.g., Random Forest)	Boosting (e.g., AdaBoost, Gradient Boosting)
Training	Parallel	Sequential (iterative)
Data Usage	Each model trained on a bootstrap sample (random subset with replacement).	Each new model focuses on data points misclassified by previous models (weighted data).
Model Weighting	All base models typically have equal weight in final prediction.	Models are weighted based on their performance; misclassified data points are weighted.
Primary Goal	**Reduce Variance (address overfitting)**	**Reduce Bias (address underfitting, but also impacts variance)**
Base Learners	Can use strong, complex learners (e.g., deep decision trees).	Typically uses weak, simple learners (e.g., shallow decision trees/stumps).
Overfitting Risk	Less prone to overfitting.	More prone to overfitting if not carefully tuned (especially with noisy data).
Speed	Faster (due to parallelization).	Slower (due to sequential nature).
Robustness	More robust to noise and outliers.	More sensitive to noise and outliers.

Random Subspace:

- P adet değişken (100) var diyelim dallanmalara karar verilen yerde random subspace ile seçilen 20 değişkenle dallanmaya başlanılır

Python Application:

n_estimators:

- Nedir: Ormandaki (ensemble) karar ağaçlarının sayısı.
- Etkisi: Daha fazla ağaç genellikle daha iyi performansa yol açar, ancak hesaplama maliyetini artırır ve belirli bir noktadan sonra ek fayda sağlamaz.

criterion:

- **Nedir:** Bir düğümde en iyi ayrımı (split) seçmek için kullanılan fonksiyon. Genellikle "gini" (Gini belirsizliği) veya "entropy" (bilgi kazancı) seçenekleri

kullanılır.

- **Etkisi:** İki ölçüt de genellikle benzer sonuçlar verir. "Entropy" biraz daha yoğun hesaplamalı olabilir.

max_depth:

- **Nedir:** Her bir ağacın izin verilen maksimum derinliği. `None` ise düğümler tamamen genişletilir.

min_samples_split:

- **Nedir:** Bir düğümü bölmek için gereken minimum örnek sayısı.
- **Etkisi:** Bu değer ne kadar büyük olursa, ağaç o kadar az dallanır ve overfitting riski o kadar azalır.

min_samples_leaf (default→1)

- **Nedir:** Bir yaprak düğümde (terminal düğüm) bulunması gereken minimum örnek sayısı.

min_weight_fraction_leaf

max_features="sqrt"

- **Nedir:** Her bir ayrım noktasında (split) aday olarak değerlendirilecek maksimum özellik sayısı. "sqrt" (varsayılan) genellikle özellik sayısının karekökünü kullanır, "log2" logaritmasını kullanır, `None` tüm özellikleri kullanır.
- **Etkisi:** Random Forest'ın "rastgele" tarafının anahtarıdır. Özelliklerin rastgele seçilmesi, ağaçların birbirinden daha farklı olmasını sağlar ve korelasyonu azaltarak varyansı düşürür.

max_leaf_nodes=None

min_impurity_decrease=0.0

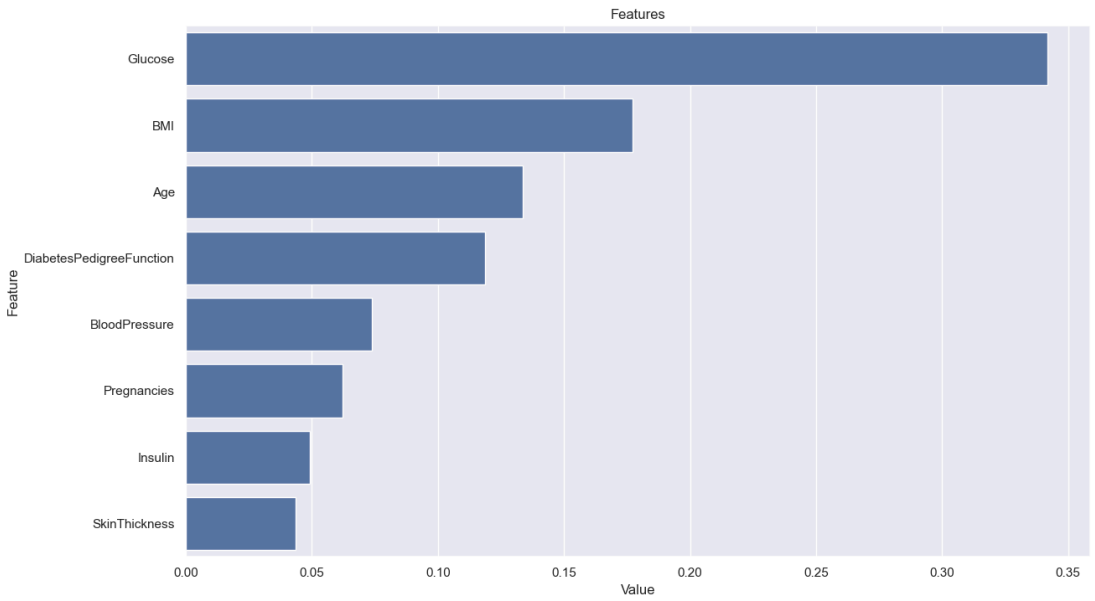
bootstrap=True oob_score=False,

n_jobs=None

- **Nedir:** Eğitimi ve tahmini paralel olarak çalıştırmak için kullanılacak CPU çekirdeği sayısı. `1` tüm mevcut çekirdekleri kullanır.
- **Etkisi:** Büyük modeller ve/veya büyük veri setleri için eğitim süresini önemli ölçüde hızlandırabilir.

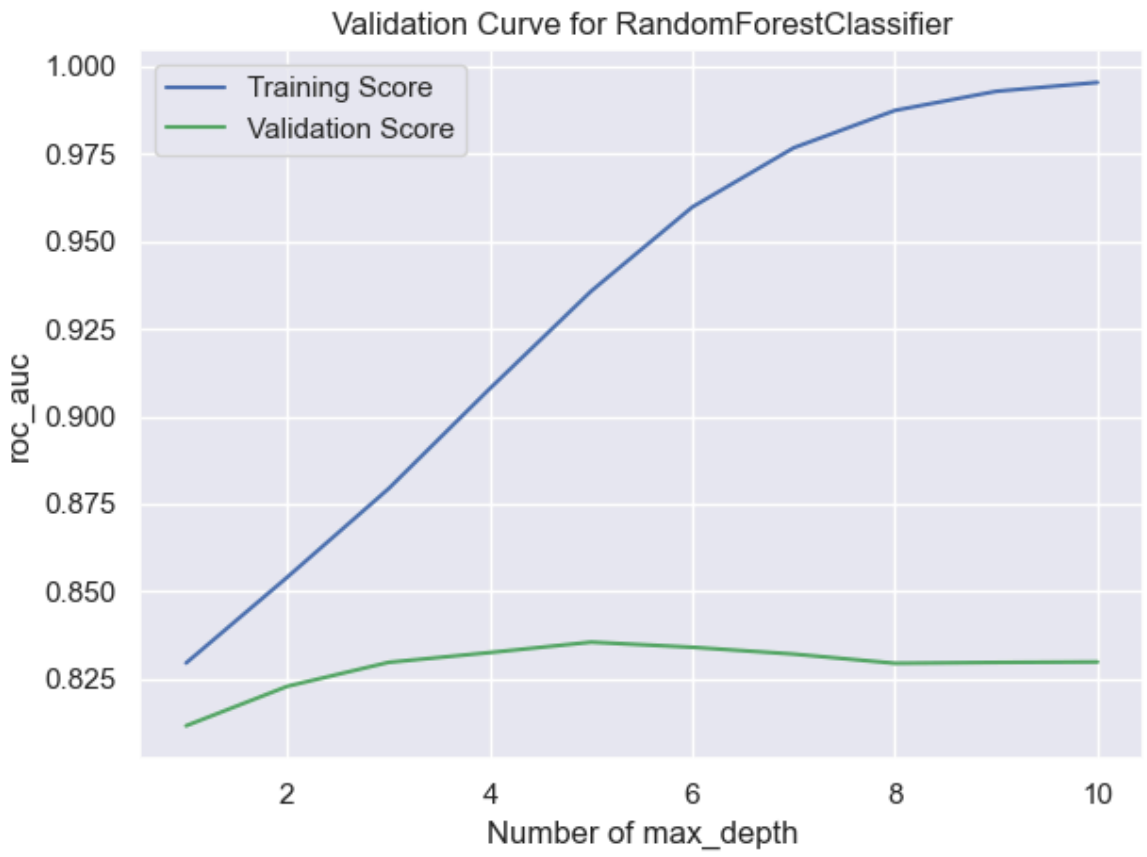
```
##################################################
# Random Forests
##################################################

rf_model = RandomForestClassifier(random_state=17)
rf_model.get_params()

cv_results = cross_validate(rf_model, X, y, cv=10, scoring=["accuracy", "f1",
"roc_auc"])
cv_results['test_accuracy'].mean() # np.float64(0.753896103896104)
cv_results['test_f1'].mean() #np.float64(0.6190701534636385)
cv_results['test_roc_auc'].mean() #np.float64(0.8233960113960114)


rf_params = {"max_depth": [5, 8, None],
             "max_features": [3, 5, 7, "auto"],
             "min_samples_split": [2, 5, 8, 15, 20],
             "n_estimators": [100, 200, 500]}

#Toplam 900 fit etme
rf_best_grid = GridSearchCV(rf_model, rf_params, cv=5, n_jobs=-1, verbos
e=True).fit(X, y)

rf_best_grid.best_params_
#{'max_depth': None, 'max_features': 5, 'min_samples_split': 8, 'n_estimato
rs': 500}

rf_final = rf_model.set_params(**rf_best_grid.best_params_, random_state
=17).fit(X, y)

cv_results = cross_validate(rf_final, X, y, cv=10, scoring=["accuracy", "f1",
"roc_auc"])
cv_results['test_accuracy'].mean() #np.float64(0.766848940533151)
cv_results['test_f1'].mean() #np.float64(0.6447777811143756)
cv_results['test_roc_auc'].mean() #np.float64(0.8271054131054132)
```

```
def plot_importance(model, features, num=len(X), save=False):
    feature_imp = pd.DataFrame({'Value': model.feature_importances_, 'Feat
ure': features.columns})
    plt.figure(figsize=(10, 10))
    sns.set(font_scale=1)
    sns.barplot(x="Value", y="Feature", data=feature_imp.sort_values(by="V
alue",
                                                     ascending=False)[0:num])
    plt.title('Features')
    plt.tight_layout()
    plt.show()
    if save:
        plt.savefig('importances.png')

plot_importance(rf_final, X)
```




```
def val_curve_params(model, X, y, param_name, param_range, scoring="ro
c_auc", cv=10):
    train_score, test_score = validation_curve(
        model, X=X, y=y, param_name=param_name, param_range=param_ra
nge, scoring=scoring, cv=cv)
```

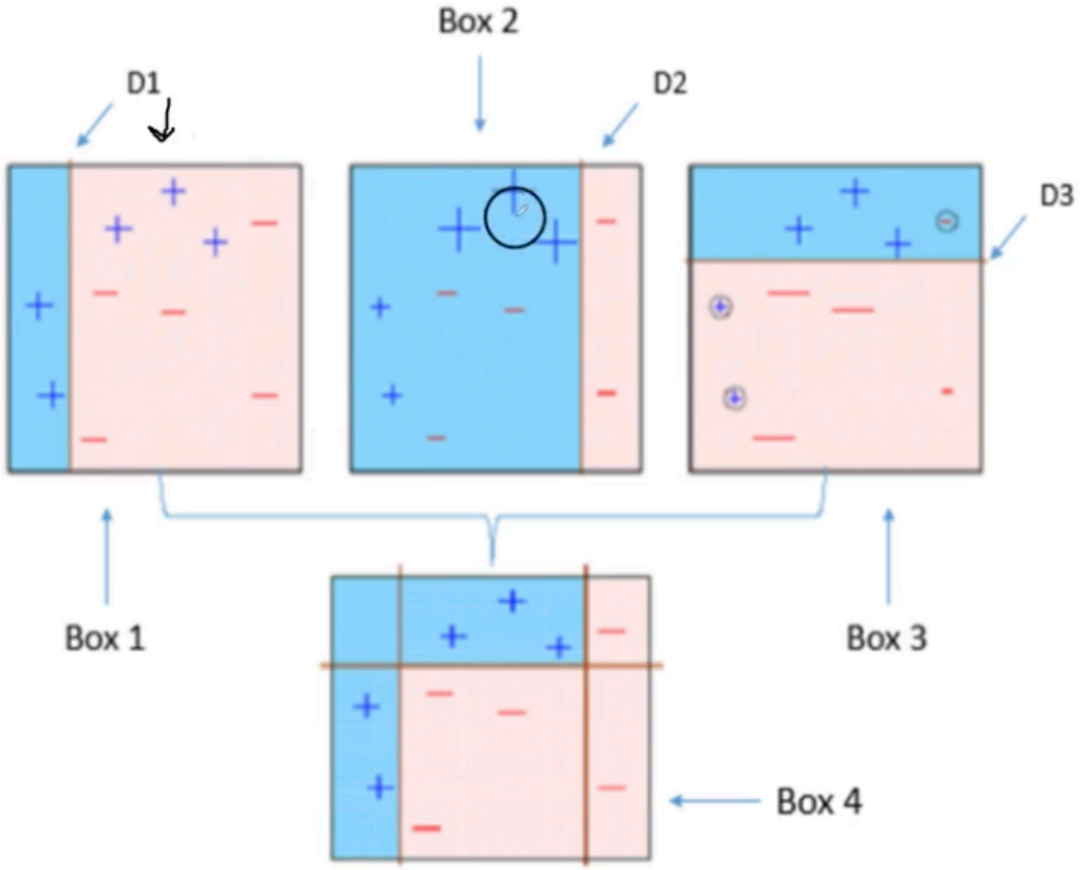
```
    mean_train_score = np.mean(train_score, axis=1)
    mean_test_score = np.mean(test_score, axis=1)

    plt.plot(param_range, mean_train_score,
             label="Training Score", color='b')

    plt.plot(param_range, mean_test_score,
             label="Validation Score", color='g')

    plt.title(f"Validation Curve for {type(model).__name__}")
    plt.xlabel(f"Number of {param_name}")
    plt.ylabel(f"{scoring}")
    plt.tight_layout()
    plt.legend(loc='best')
    plt.show(block=True)

val_curve_params(rf_final, X, y, "max_depth", range(1, 11), scoring="roc_au
c")
```



Gradient Boosting Machines (GBM)-GBT

- **Adaptive Boosting (AdaBoost)**, bir **Boosting** algoritmasıdır ve makine öğrenmesinde sıkça kullanılan bir **ensemble (topluluk) öğrenme** yöntemidir. Temel amacı, birden fazla "zayıf öğreniciyi" (weak learner - örneğin, tek bir karara dayalı çok basit karar ağaçları, yani "karar güdükleri" - decision stumps) bir araya getirerek çok daha güçlü ve doğru bir model oluşturmaktır.

    1. **Sıralı Eğitim:** AdaBoost, zayıf öğrenicileri **sıralı (ardışık)** bir şekilde eğitir. Her yeni öğrenici, kendinden önceki öğrenicilerin yaptığı hataları düzeltmeye odaklanır.
    2. **Örnek Ağırlıklandırması:** Algoritma başlangıçta tüm eğitim verisi örneklerine eşit ağırlık verir. Her bir zayıf öğrenici eğitildikten sonra, **yanlış sınıflandırdığı örneklere daha yüksek ağırlık verir**, doğru sınıflandırdığı örneklere ise ağırlıklarını azaltır.
    3. **Hatalara Odaklanma:** Bu ağırlıklandırma mekanizması sayesinde, sonraki zayıf öğreniciler, önceki modellerin zorlandığı veya yanlış tahmin ettiği örneklere daha fazla "dikkat" eder.
    4. **Ağırlıklı Oybirliği:** Son olarak, tüm zayıf öğreniciler birleştirilir. Her bir zayıf öğrenicinin nihai tahmine katkısı, kendi doğruluk performansına göre ağırlıklandırılır (daha doğru olanlar daha fazla söz sahibi olur).



- **Gradient Boosting Machines (GBM)**, makine öğrenmesindeki en güçlü ve yaygın olarak kullanılan ensemble (topluluk) öğrenme tekniklerinden biridir. Adı üzerinde, "boosting" ve "gradyan inişi" (gradient descent) kavramlarını bir araya getirir. Özellikle Kaggle yarışmaları gibi birçok veri bilimi rekabetinde yüksek başarı elde etmesiyle bilinir.
    - **Boosting + Gradient Descent**
    - Gradient boosting tek bir tahminsel model formunda olan modeller serisi oluşturur.
    - Seri içerisindeki bir model serideki bir önceki modelin tahmin artıklarını/ hatalarının (residuals) üzerine kurularak (fit) oluşturulur.
    - GBM diferansiyellenebilen herhangi bir kayıp fonksiyonunu optimize edebilen Gradient descent algoritmasını kullanmaktadır.

- Tek bir tahminsel model formunda olan modeller serisi **additive**, sekilde kurulur.

- **Temel Prensip: Hataları Gradyan İnişi ile Düzeltme**

  GBM'nin ana fikri, **ardışık olarak bir dizi zayıf öğreniciyi (weak learners)** (genellikle sığ karar ağaçları, yani "karar güdükleri") eğitmek ve her yeni öğreniciyi, önceki modellerin yaptığı **hataları (residuals) veya bu hataların gradyanlarını** minimize edecek şekilde inşa etmektir.

  Adım adım çalışma prensibi şöyledir:

  1. **Başlangıç Tahmini (Initial Prediction):** Model, veri setindeki hedef değişkenin (örneğin regresyon için ortalama, sınıflandırma için log-olasılık) basit bir sabit değeriyle başlar. Bu, ilk "zayıf öğrenicidir".
  2. **Hataların Hesaplanması (Calculate Residuals/Gradients):**
     - Regresyon problemlerinde, mevcut modelin gerçek değerler ile tahminleri arasındaki **kalıntıları (residuals)** hesaplanır. Bu kalıntılar, modelin henüz açıklayamadığı "hatalardır".
     - Sınıflandırma gibi daha genel kayıp fonksiyonları için ise, bu kalıntılar doğrudan değil, kayıp fonksiyonunun mevcut tahminlere göre **negatif gradyanları (pseudo-residuals)** olarak düşünülür. Bu gradyanlar, modelin hangi yönde ilerlemesi gerektiğini gösteren "hataların yönü" gibidir.
  3. **Yeni Bir Zayıf Öğrenici Eğitme:** Yeni bir zayıf öğrenici (örneğin, küçük bir karar ağacı), **bağımsız değişkenleri (özellikler)** kullanarak **önceki adımda hesaplanan kalıntıları/gradyanları tahmin etmek** üzere eğitilir. Yani, model artık doğrudan orijinal hedef değişkeni değil, **hatanın kendisini** öğrenmeye çalışır.
  4. **Model Güncelleme:** Yeni eğitilen zayıf öğrenicinin tahmini, **küçük bir öğrenme oranı (learning rate - η) ile çarpılarak** mevcut topluluk modeline eklenir. Öğrenme oranı, her yeni ağacın toplam modele ne kadar katkıda bulunacağını kontrol eder ve aşırı öğrenmeyi (overfitting) önlemeye yardımcı olur.

     $F_m(x) = F_{m-1}(x) + \eta \cdot h_m(x)$

     Burada:

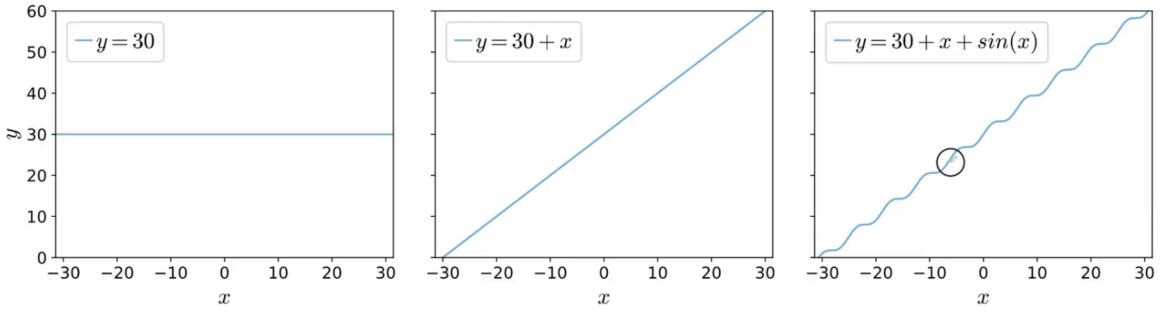
     - $F_m(x)$: m. adımdaki ensemble modelinin tahmini

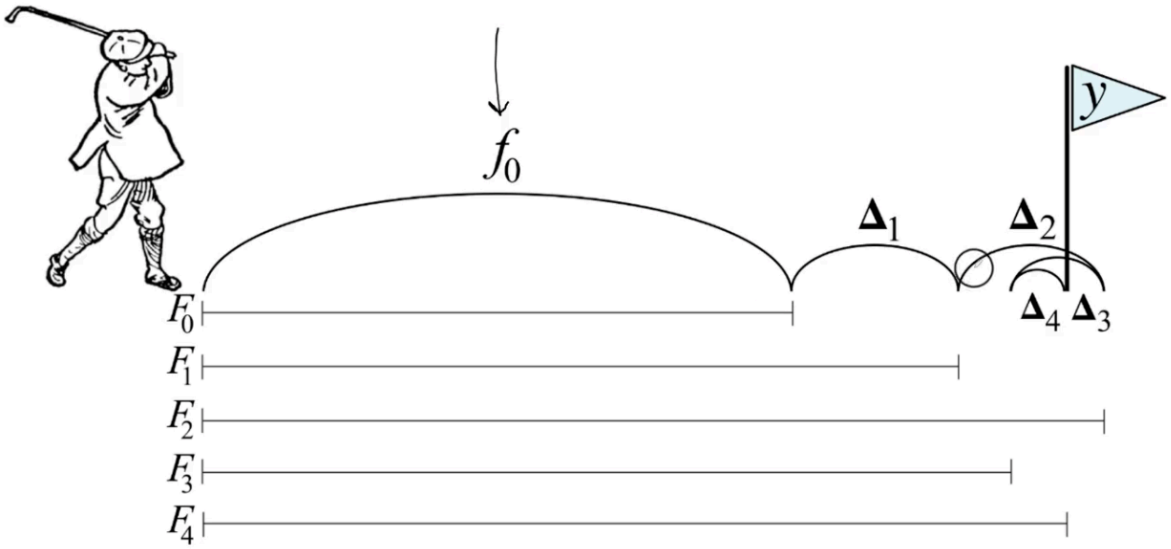
- Fm−1(x): Önceki adımdaki ensemble modelinin tahmini
- η: Öğrenme oranı
- hm(x): m. adımda eğitilen zayıf öğrenici (kalıntıları/gradyanları tahmin eden model)

5. **İterasyon (Tekrarlama):** Bu adımlar, belirli bir ağaç sayısına ulaşana veya model performansı artık iyileşmeyene kadar tekrarlanır. Her iterasyonda, model önceki hatalarını daha iyi anlamak ve düzeltmek için "öğrenir".

- **Additive Modelleme:**
  - **Additive Modelleme**, bir bağımlı değişken ile bir veya daha fazla bağımsız değişken arasındaki ilişkiyi, her bir bağımsız değişkenin bağımlı değişken üzerindeki **bireysel ve ayrı ayrı katkılarının toplamı** olarak ifade eden bir istatistiksel modelleme yaklaşımıdır.

1. **Her Değişkenin Bağımsız Katkısı:** Additive modellerin en önemli özelliği, her bir bağımsız değişkenin (X) bağımlı değişken (Y) üzerindeki etkisinin **diğer değişkenlerden bağımsız olarak** ekleniyor olmasıdır. Yani, X1'in etkisi, X2'nin değerine bağlı olarak değişmez. Bu, modelin yorumlanmasını oldukça kolaylaştırır, çünkü her bir fj(Xj) fonksiyonunun grafiğine bakarak ilgili Xj değişkeninin Y üzerindeki etkisini doğrudan anlayabiliriz.
2. **Doğrusallık Kısıtının Kaldırılması:** Geleneksel çoklu doğrusal regresyon modelleri, bağımsız değişkenler ile bağımlı değişken arasında doğrusal bir ilişki olduğunu varsayar. Additive modeller ise bu doğrusal kısıtlamayı kaldırır ve fj(Xj) fonksiyonları aracılığıyla **doğrusal olmayan ilişkileri de yakalayabilir**. Bu fonksiyonlar genellikle veriyle uyumlu, "yumuşak" (smooth) eğriler olarak tahmin edilir.
3. **Yorumlanabilirlik ve Esneklik Dengesi:** Additive modeller, bir yandan doğrusal modellerin basit yorumlanabilirlik avantajını korurken (her değişkenin katkısı ayrı ayrı incelenebilir), diğer yandan daha karmaşık, doğrusal olmayan ilişkileri modelleyebilme esnekliği sunar. Bu yönüyle "kara kutu" (black-box) makine öğrenmesi modelleri ile basit doğrusal modeller arasında bir köprü görevi görürler.

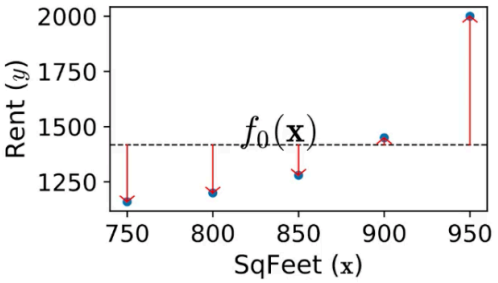




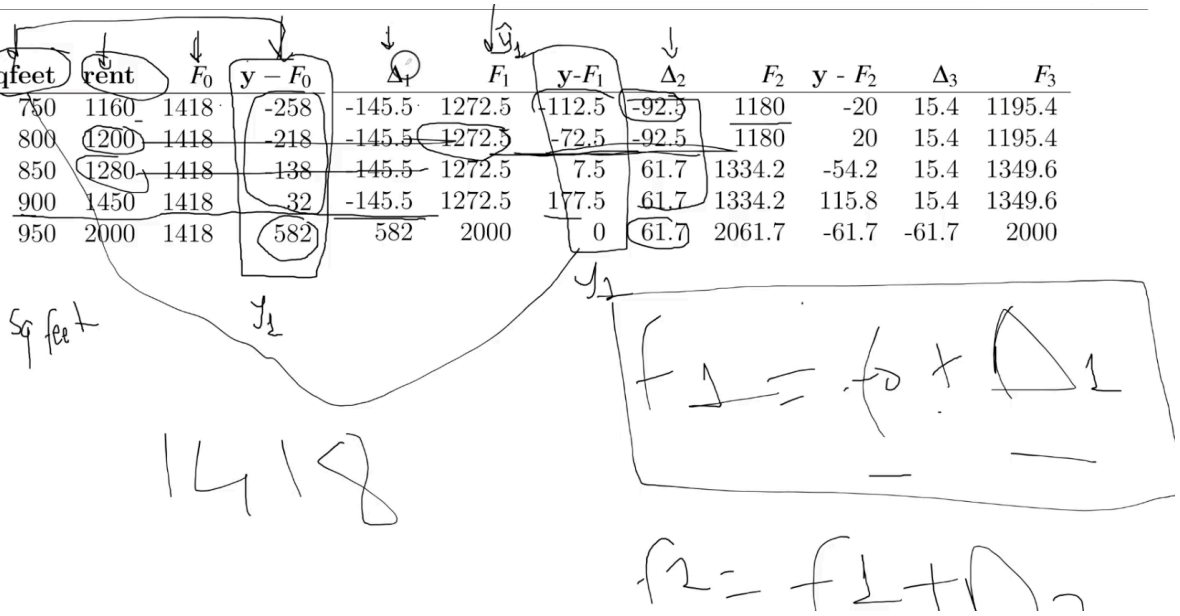
$$
\begin{aligned}
\hat{y} &= f_0(\mathbf{x}) + \Delta_1(\mathbf{x}) + \Delta_2(\mathbf{x}) + \ldots + \Delta_M(\mathbf{x}) \\
&= f_0(\mathbf{x}) + \sum_{m=1}^{M} \Delta_m(\mathbf{x}) \\
&= F_M(\mathbf{x})
\end{aligned}
$$

$$
\begin{aligned}
F_0(\mathbf{x}) &= f_0(\mathbf{x}) \\
F_m(\mathbf{x}) &= F_{m-1}(\mathbf{x}) + \Delta_m(\mathbf{x})
\end{aligned}
$$

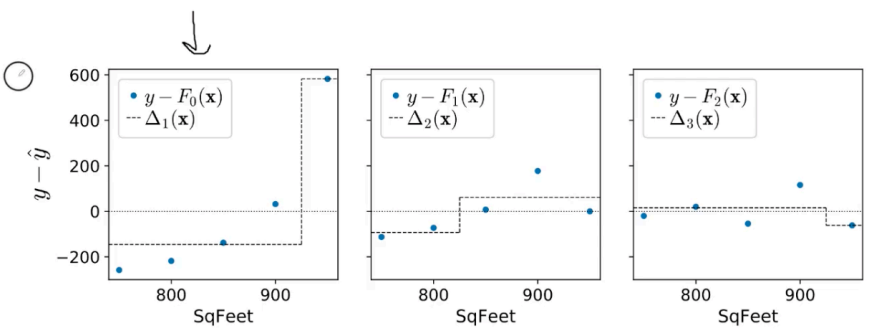
sqfeet	rent	F_0	$y - F_0$
750	1160	1418	-258
800	1200	1418	-218
850	1280	1418	-138
900	1450	1418	32
950	2000	1418	582



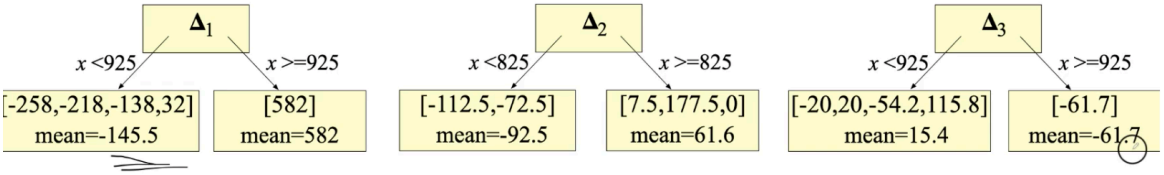
sqfeet	rent	F_0	$y - F_0$	Δ_1	F_1	y-F_1	Δ_2	F_2	$y - F_2$	Δ_3	F_3
750	1160	1418	-258	-145.5	1272.5	-112.5	-92.5	1180	-20	15.4	1195.4
800	1200	1418	-218	-145.5	1272.5	-72.5	-92.5	1180	20	15.4	1195.4
850	1280	1418	-138	-145.5	1272.5	7.5	61.7	1334.2	-54.2	15.4	1349.6
900	1450	1418	32	-145.5	1272.5	177.5	61.7	1334.2	115.8	15.4	1349.6
950	2000	1418	582	582	2000	0	61.7	2061.7	-61.7	-61.7	2000



sqfeet	rent	F_0	$y - F_0$	Δ_1	F_1	y-F_1	Δ_2	F_2	$y - F_2$	Δ_3	F_3
750	1160	1418	-258	-145.5	1272.5	-112.5	-92.5	1180	-20	15.4	1195.4
800	1200	1418	-218	-145.5	1272.5	-72.5	-92.5	1180	20	15.4	1195.4
850	1280	1418	-138	-145.5	1272.5	7.5	61.7	1334.2	-54.2	15.4	1349.6
900	1450	1418	32	-145.5	1272.5	177.5	61.7	1334.2	115.8	15.4	1349.6
950	2000	1418	582	582	2000	0	61.7	2061.7	-61.7	-61.7	2000



sqfeet	rent	F_0	$y - F_0$	Δ_1	F_1	$y - F_1$	Δ_2	F_2	$y - F_2$	Δ_3	F_3
750	1160	1418	-258	-145.5	1272.5	-112.5	-92.5	1180	-20	15.4	1195.4
800	1200	1418	-218	-145.5	1272.5	-72.5	-92.5	1180	20	15.4	1195.4
850	1280	1418	-138	-145.5	1272.5	7.5	61.7	1334.2	-54.2	15.4	1349.6
900	1450	1418	32	-145.5	1272.5	177.5	61.7	1334.2	115.8	15.4	1349.6
950	2000	1418	582	582	2000	0	61.7	2061.7	-61.7	-61.7	2000



GBM Uygulaması

- *References:* *J. Friedman, Greedy Function Approximation: A Gradient BoostingMachine, The Annals of Statistics, Vol. 29, No. 5, 2001.J. Friedman, Stochastic Gradient Boosting, 1999T. Hastie, R. Tibshirani and J. Friedman.Elements of Statistical Learning Ed. 2, Springer, 2009.*

Aşağıdaki parametreler, Gradient Boosting modelinin öğrenme sürecini ve karmaşıklığını kontrol eder. En iyi performans için dikkatli bir şekilde ayarlanmaları (tuning) gerekir.

- **learning_rate** : [0.01, 0.1]
  - **Nedir:** Her bir ağacın (weak learner) toplam modele ne kadar katkıda bulunacağını belirleyen bir ağırlık faktörüdür.
  - **Etkisi:** Düşük bir öğrenme oranı, modelin hataları daha küçük adımlarla düzeltmesini sağlar, bu da genellikle daha sağlam ve genellenebilir bir modelle sonuçlanır ancak daha fazla n_estimators (ağaç sayısı) gerektirir ve eğitim süresini uzatır. Yüksek bir öğrenme oranı ise daha hızlı öğrenebilir ama **aşırı uyuma (overfitting) daha** yatkın hale getirebilir. Burada 0.01 ve 0.1 değerleri, modelin öğrenme hızını test etmek için belirlenmiş.
- **max_depth** : [3, 8, 10]

    - **Nedir: Her bir bireysel karar ağacının (weak learner) izin verilen maksimum derinliği.**
- `n_estimators` : `[100, 500, 1000]`
    - **Nedir:** Boosting sürecinde oluşturulacak (yani ensemble'a dahil edilecek) karar ağaçlarının toplam sayısı.
    - The number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance.
- `subsample` : `[1, 0.5, 0.7]`
    - **Nedir:** Her bir bireysel ağaç eğitilirken, eğitim verisinin ne kadarının (oran olarak) kullanılacağını belirler. `1` olması, tüm veri setinin kullanılacağı anlamına gelirken, `0.5` %50'sinin kullanılacağı anlamına gelir. Bu örnekleme **yerine koymasız** yapılır.
- `gbm_final = gbm_model.set_params(**gbm_best_params_, random_state=17).fit(X, y)` :
    - Burada ise, `GridSearchCV` ile bulunan en iyi parametreler (`gbm_best_params_`) `gbm_model`'e uygulanıyor ve ardından model `X` ve `y` üzerinde yeniden eğitiliyor.
    - Bu aşamada `random_state=17`'nin tekrar atanmasının **iki temel nedeni** olabilir:
        - **Tutarlılık:** Modelin ilk tanımlandığındaki rastgele durumunu korumak ve `set_params` ile olası bir sıfırlanmayı veya değişimi engellemek. Bu, modelin tüm eğitim süreçleri boyunca aynı temel rastgeleliği kullanmasını garanti eder.
        - **Emin Olma:** Bazı durumlarda `set_params` metodu, modelin tüm içsel durumunu (random state dahil) doğrudan optimize edilen parametrelerle birlikte güncelleyebilir veya varsayılana döndürebilir. `random_state=17`'nin açıkça tekrar belirtilmesi, bu parametrenin kesinlikle belirlenen değerde kalmasını ve nihai modelin eğitiminin de tamamen tekrarlanabilir olmasını sağlar.

```
##################################################
# GBM
##################################################
```

```
gbm_model = GradientBoostingClassifier(random_state=17)

gbm_model.get_params()

cv_results = cross_validate(gbm_model, X, y, cv=5, scoring=["accuracy", "f
1", "roc_auc"])
cv_results['test_accuracy'].mean()
# 0.7591715474068416
cv_results['test_f1'].mean()
# 0.634
cv_results['test_roc_auc'].mean()
# 0.82548

gbm_params = {"learning_rate": [0.01, 0.1],
              "max_depth": [3, 8, 10],
              "n_estimators": [100, 500, 1000],
              "subsample": [1, 0.5, 0.7]}

gbm_best_grid = GridSearchCV(gbm_model, gbm_params, cv=5, n_jobs=-
1, verbose=True).fit(X, y)

gbm_best_grid.best_params_
#{'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 1000, 'subsample': 0.
7}
gbm_final = gbm_model.set_params(**gbm_best_grid.best_params_, rando
m_state=17, ).fit(X, y)



cv_results = cross_validate(gbm_final, X, y, cv=5, scoring=["accuracy", "f1",
"roc_auc"])
cv_results['test_accuracy'].mean() #np.float64(0.7760886172650878)
cv_results['test_f1'].mean() #np.float64(0.6645104311829364)
cv_results['test_roc_auc'].mean() #np.float64(0.83546750524109)
```

XGBoost (eXtreme Gradient Boosting)

- XGBoost, Gradient Boosting'in gücünü alıp, onu **hız, ölçeklenebilirlik, düzenlileştirme ve esneklik** açısından ciddi şekilde iyileştiren bir algoritmadır. Bu özellikler sayesinde, birçok veri bilimci ve makine öğrenimi uygulamasında tercih edilen bir araç haline gelmiştir.
- **XGBoost**, Gradient Boosting Machine (GBM) algoritmasının optimize edilmiş, ölçeklenebilir ve yüksek performanslı bir dağıtımıdır. Chen ve Guestrin tarafından 2016 yılında bilimsel bir makalede tanıtılmıştır. Çok çeşitli makine öğrenimi görevlerinde (sınıflandırma, regresyon, sıralama vb.) rekabetçi performansıyla öne çıkar ve özellikle tabular verilerde sıkça tercih edilir.
- XGBoost, temel olarak **Gradyan Yükseltme (Gradient Boosting)** prensiplerine dayanır, yani zayıf öğrenicileri (genellikle karar ağaçları) sıralı bir şekilde eğiterek, önceki modellerin hatalarını (rezidüellerini veya kayıp fonksiyonunun gradyanlarını) düzeltmeye çalışır. Ancak bunu yaparken, bazı önemli optimizasyonlar ve iyileştirmeler getirir.
- **Nasıl çalışır:**
    1. Model, tüm eğitim verisi için basit bir başlangıç tahminiyle (örneğin, tüm örneklerin ortalaması) başlar.
    2. Her iterasyonda, mevcut modelin gerçek değerler ile tahminler arasındaki **hataları (rezidüelleri)** veya **kayıp fonksiyonunun gradyanlarını ve Hessian'ını** hesaplar.
    3. Yeni bir karar ağacı (genellikle sığ bir ağaç), bu **hataları tahmin etmek** üzere eğitilir. Ağacın yapılandırılmasında da düzenlileştirme terimleri dikkate alınır.
    4. Yeni ağacın tahmini, bir **öğrenme oranı** (learning rate) ile çarpılarak mevcut modele eklenir. Öğrenme oranı, her yeni ağacın katkısını kontrol ederek aşırı öğrenmeyi önlemeye yardımcı olur.
    5. Bu süreç, belirli bir ağaç sayısına ulaşana veya erken durdurma kriteri karşılanana kadar tekrarlanır.

```
##################################################
# XGBoost
##################################################

xgboost_model = XGBClassifier(random_state=17, use_label_encoder=Fals
```

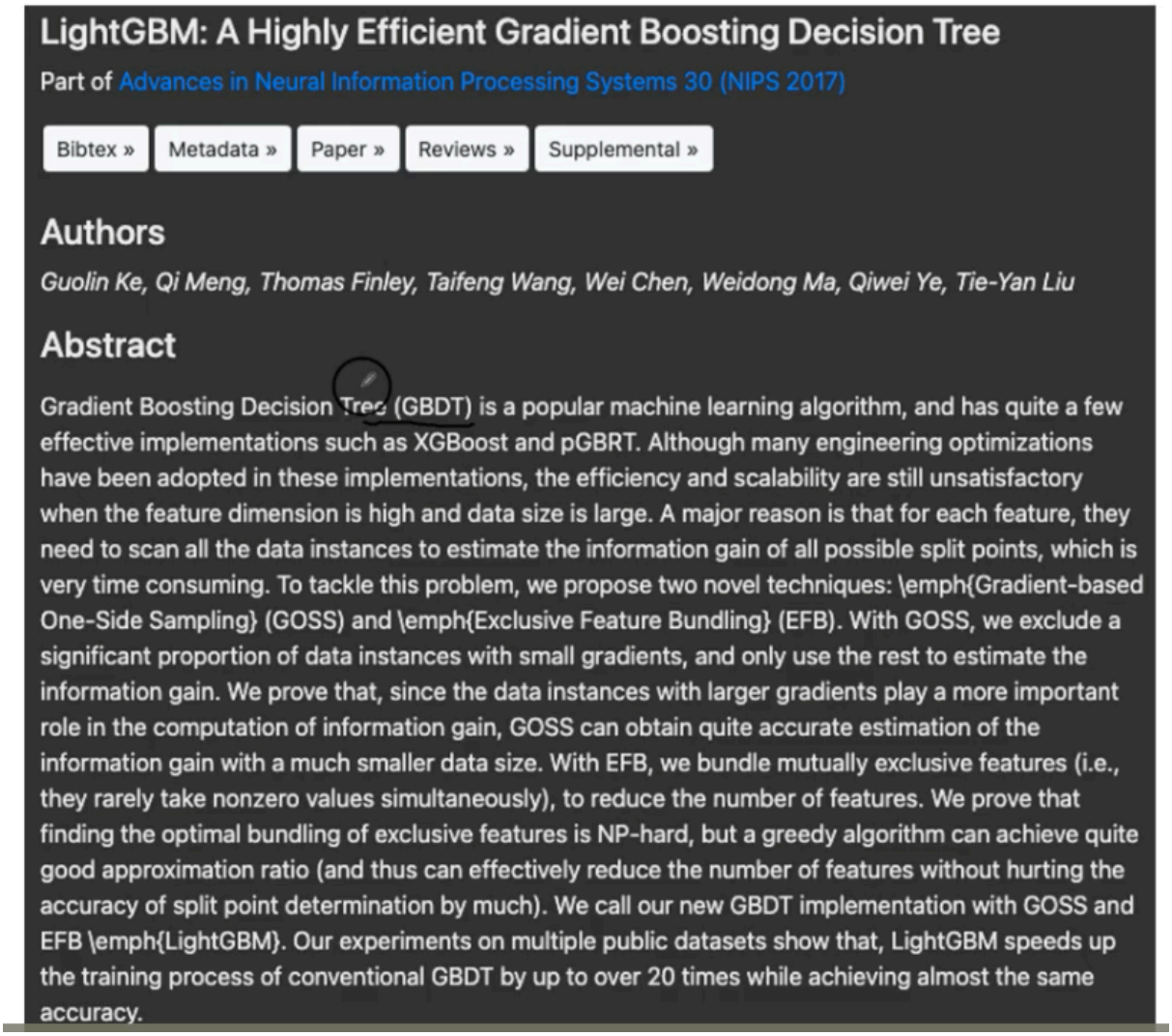
```
e)
xgboost_model.get_params()
cv_results = cross_validate(xgboost_model, X, y, cv=5, scoring=["accurac
y", "f1", "roc_auc"])
cv_results['test_accuracy'].mean()
# 0.75265
cv_results['test_f1'].mean()
# 0.631
cv_results['test_roc_auc'].mean()
# 0.7987

#colsample_bytree: This is a family of parameters for subsampling of colu
mns.

xgboost_params = {"learning_rate": [0.1, 0.01],
                  "max_depth": [5, 8],
                  "n_estimators": [100, 500, 1000],
                  "colsample_bytree": [0.7, 1]}

xgboost_best_grid = GridSearchCV(xgboost_model, xgboost_params, cv=
5, n_jobs=-1, verbose=True).fit(X, y)
#Hyperparametre optimizasyonu
xgboost_final = xgboost_model.set_params(**xgboost_best_grid.best_para
ms_, random_state=17).fit(X, y)
#{'colsample_bytree': 0.7, 'learning_rate': 0.1, 'max_depth': 5, 'n_estimator
s': 100}

cv_results = cross_validate(xgboost_final, X, y, cv=5, scoring=["accuracy",
"f1", "roc_auc"])
cv_results['test_accuracy'].mean() #np.float64(0.7617689500042442)
cv_results['test_f1'].mean() #np.float64(0.6363347763347763)
cv_results['test_roc_auc'].mean()#np.float64(0.8183675751222921)
```

Özellik	**Genel GBM (Traditionel Uygulamalar)**	**XGBoost (eXtreme Gradient Boosting)**
Temel Felsefe	Hataları (rezidüelleri) veya kayıp fonksiyonunun negatif gradyanlarını	Temel GBM prensibi aynı, ancak performans ve

	düzelterek zayıf öğrenicileri (ağaçları) sıralı olarak ekleme.	özellik seti açısından optimize edilmiş.
Ağaç İnşası	Genellikle ağaçları sırayla ve daha basit yöntemlerle oluşturur.	Ağaç inşasını paralel hale getirir (düğüm bölünmeleri için).
Kayıp Fonk. Opt.	Kayıp fonksiyonunun sadece birinci dereceden türevlerini (gradyanlarını) kullanır.	Kayıp fonksiyonunun birinci ve ikinci dereceden türevlerini (Hessian) kullanarak daha doğru ve hızlı optimizasyon.
Düzenlileştirme (Regularization)	Genellikle sadece learning_rate, n_estimators, max_depth gibi parametrelerle dolaylı düzenlileştirme.	Doğrudan L1 (Lasso) ve L2 (Ridge) düzenlileştirme terimlerini kayıp fonksiyonuna ekler. Ağaç budama da daha etkindir.
Kayıp Veri Yönetimi	Genellikle kayıp değerleri özel ön işlem gerektirir.	Kayıp değerleri (NaN'lar) otomatik olarak ele alır ve en iyi yolu öğrenir.
Seyrek Veri Desteği	Genellikle seyrek veriler için özel optimizasyonlara sahip değildir.	Seyrek veriler (sparse-aware) için optimize edilmiş algoritmalar kullanır.
Bellek Optimizasyonu	Bellek kullanımı daha az optimize edilmiş olabilir.	Veri blokları ve sıkıştırma yöntemleri ile daha verimli bellek kullanımı.
Erken Durdurma (Early Stopping)	Tüm GBM uygulamalarında her zaman bulunmayabilir veya daha temel seviyede olabilir.	Dahili olarak erken durdurma özelliği sunar, bu da aşırı öğrenmeyi önler ve eğitim süresini kısaltır.
Performans	Sağlam ve iyi bir performans sağlar.	Özellikle büyük ve karmaşık veri setlerinde daha hızlı ve daha yüksek doğruluk sunar.
Kullanım Alanı	Çeşitli ML görevleri için uygun.	Kaggle yarışmaları ve endüstriyel uygulamalarda state-of-the-art performansıyla sıkça tercih edilir.
Uygulamalar	Scikit-learn'deki GradientBoostingClassifier/Regressor gibi.	XGBoost kütüphanesi (Python, R, Java, C++ vb. diller için).

LightGBM

- **LightGBM (Light Gradient Boosting Machine)**, Microsoft tarafından geliştirilen, **yüksek performanslı ve ölçeklenebilir** bir Gradient Boosting Framework'üdür. XGBoost gibi, bu da Gradient Boosting Machines (GBM) ailesinin bir üyesidir, ancak özellikle **büyük veri setleri üzerinde çok daha hızlı eğitim süreleri** sunmak ve daha az bellek kullanmak üzere tasarlanmıştır.
- LightGBM, XGBoost'un eğitim süresi performansını arttırmaya yönelik geliştirilen bir diğer GBM türüdür.
- Level-wise büyüme stratejisi yerine Leaf-wise büyüme stratejisi ile daha hızlıdır.



LightGBM: A Highly Efficient Gradient Boosting Decision Tree

Part of Advances in Neural Information Processing Systems 30 (NIPS 2017)

Bibtex » Metadata » Paper » Reviews » Supplemental »

Authors

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, Tie-Yan Liu

Abstract

Gradient Boosting Decision Tree (GBDT) is a popular machine learning algorithm, and has quite a few effective implementations such as XGBoost and pGBRT. Although many engineering optimizations have been adopted in these implementations, the efficiency and scalability are still unsatisfactory when the feature dimension is high and data size is large. A major reason is that for each feature, they need to scan all the data instances to estimate the information gain of all possible split points, which is very time consuming. To tackle this problem, we propose two novel techniques: \emph{Gradient-based One-Side Sampling} (GOSS) and \emph{Exclusive Feature Bundling} (EFB). With GOSS, we exclude a significant proportion of data instances with small gradients, and only use the rest to estimate the information gain. We prove that, since the data instances with larger gradients play a more important role in the computation of information gain, GOSS can obtain quite accurate estimation of the information gain with a much smaller data size. With EFB, we bundle mutually exclusive features (i.e., they rarely take nonzero values simultaneously), to reduce the number of features. We prove that finding the optimal bundling of exclusive features is NP-hard, but a greedy algorithm can achieve quite good approximation ratio (and thus can effectively reduce the number of features without hurting the accuracy of split point determination by much). We call our new GBDT implementation with GOSS and EFB \emph{LightGBM}. Our experiments on multiple public datasets show that, LightGBM speeds up the training process of conventional GBDT by up to over 20 times while achieving almost the same accuracy.

Özellik / Algoritma	**Geleneksel GBM (Scikit-learn)**	**XGBoost**	**LightGBM**
Geliştirici	Çeşitli (örneğin Friedman, 2001)	Tianqi Chen & Carlos Guestrin (University of Washington)	Microsoft
Tarih	2000'lerin başı	2014-2016	2017
Odak Noktası	Genel ensemble gücünü artırma.	Hız, performans, ölçeklenebilirlik ve düzenlileştirme.	Hız, bellek kullanımı, büyük veri setleri üzerinde performans.
Hız	Orta	Hızlı	Çok Hızlı
Bellek Kullanımı	Orta	Orta (XGBoost'a göre daha iyi)	Çok Düşük
Ağaç Büyütme	Genellikle Seviye Bazlı (Level-wise)	Seviye Bazlı (Level-wise)	Yaprak Bazlı (Leaf-wise) (varsayılan)
Düzenlileştirme	Temel parametrelerle (örn: max_depth)	L1/L2 düzenlileştirme, ağaç budama, gamma	Daha az varsayılan düzenlileştirme; max_depth, min_child_samples gibi parametrelerle yapılır.
Kayıp Veri İşleme	Genellikle ön işlem gerektirir.	Dahili (en iyi yönü öğrenir).	Dahili (NaN'ları sıfır olarak işleyebilir veya yönü öğrenir).
Paralelleşme	Kısıtlı (çoğunlukla özellik seviyesinde).	Ağaç inşasında (düğüm bölünmeleri) paralel.	Daha fazla paralel algoritma (özellik ve veri paralel).
Optimizasyonlar	Temel gradyan inişi.	Hessian kullanan 2. dereceden optimizasyon; blok yapısı, seyrek veri farkındalığı.	GSS, EFB, histogram tabanlı algoritmalar.
Aşırı Öğrenme Riski	Var	Düşük (güçlü düzenlileştirme ile).	Yüksek (küçük veri setlerinde yaprak bazlı büyüme nedeniyle), dikkatli ayar gerektirir.
Uygun Veri Boyutu	Orta-Büyük	Orta-Çok Büyük	Büyük-Çok Büyük

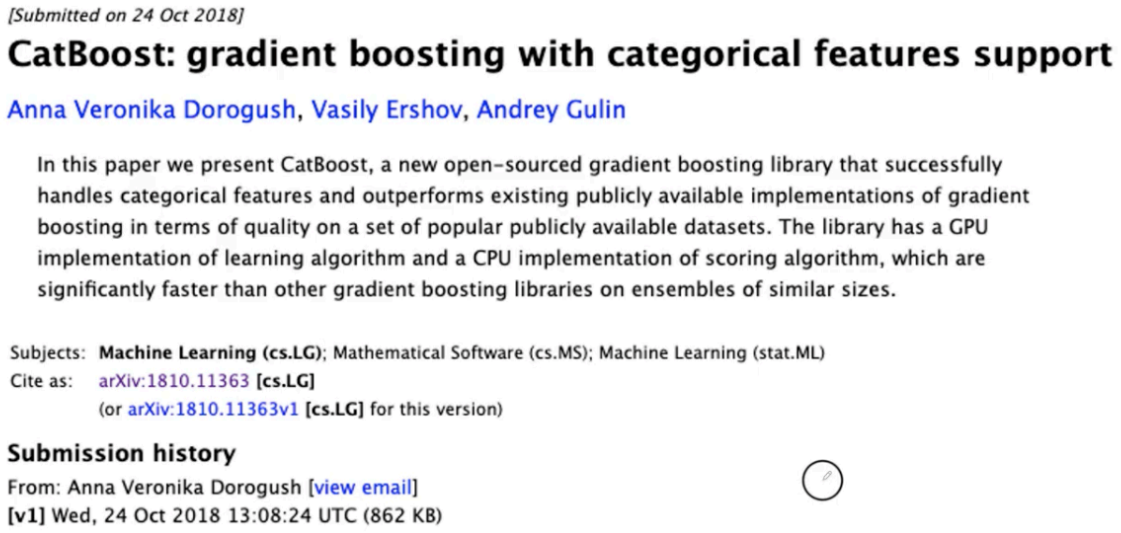
```
##################################################
# LightGBM
##################################################

lgbm_model = LGBMClassifier(random_state=17)
lgbm_model.get_params()

cv_results = cross_validate(lgbm_model, X, y, cv=5, scoring=["accuracy", "f
1", "roc_auc"])

cv_results['test_accuracy'].mean() #np.float64(0.7474492827434004)
cv_results['test_f1'].mean() #np.float64(0.624110522144179)
cv_results['test_roc_auc'].mean() #np.float64(0.7990293501048218)

lgbm_params = {"learning_rate": [0.01, 0.1],
               "n_estimators": [100, 300, 500, 1000],
               "colsample_bytree": [0.5, 0.7, 1]}

lgbm_best_grid = GridSearchCV(lgbm_model, lgbm_params, cv=5, n_jobs=
-1, verbose=True).fit(X, y)

lgbm_final = lgbm_model.set_params(**lgbm_best_grid.best_params_, rand
om_state=17).fit(X, y)

cv_results = cross_validate(lgbm_final, X, y, cv=5, scoring=["accuracy", "f
1", "roc_auc"])

cv_results['test_accuracy'].mean()
cv_results['test_f1'].mean()
cv_results['test_roc_auc'].mean()

# Hiperparametre yeni değerlerle
lgbm_params = {"learning_rate": [0.01, 0.02, 0.05, 0.1],
               "n_estimators": [200, 300, 350, 400],
               "colsample_bytree": [0.9, 0.8, 1]}

lgbm_best_grid = GridSearchCV(lgbm_model, lgbm_params, cv=5, n_jobs=
```

```
-1, verbose=True).fit(X, y)

lgbm_final = lgbm_model.set_params(**lgbm_best_grid.best_params_, rand
om_state=17).fit(X, y)

cv_results = cross_validate(lgbm_final, X, y, cv=5, scoring=["accuracy", "f
1", "roc_auc"])

cv_results['test_accuracy'].mean() #np.float64(0.7643578643578645)
cv_results['test_f1'].mean() #np.float64(0.6372062920577772)
cv_results['test_roc_auc'].mean() #np.float64(0.8147491264849755)


# Hiperparametre optimizasyonu sadece n_estimators için.
lgbm_model = LGBMClassifier(random_state=17, colsample_bytree=0.9, lea
rning_rate=0.01)

lgbm_params = {"n_estimators": [200, 400, 1000, 5000, 8000, 9000, 1000
0]}

lgbm_best_grid = GridSearchCV(lgbm_model, lgbm_params, cv=5, n_jobs=
-1, verbose=True).fit(X, y)

lgbm_final = lgbm_model.set_params(**lgbm_best_grid.best_params_, rand
om_state=17).fit(X, y)
#{'n_estimators': 200}
cv_results = cross_validate(lgbm_final, X, y, cv=5, scoring=["accuracy", "f
1", "roc_auc"])

cv_results['test_accuracy'].mean() #np.float64(0.7643833290892115)
cv_results['test_f1'].mean() #np.float64(0.6193071162618689)
cv_results['test_roc_auc'].mean() #np.float64(0.8227931516422082)
```

CatBoost:

- Kategorik değişkenler ile otomatik olarak mücadele edebilen, hızlı, başarılı bir diger GBM türevi.

- GPU desteği vardır.



[Submitted on 24 Oct 2018]

CatBoost: gradient boosting with categorical features support

Anna Veronika Dorogush, Vasily Ershov, Andrey Gulin

In this paper we present CatBoost, a new open-sourced gradient boosting library that successfully handles categorical features and outperforms existing publicly available implementations of gradient boosting in terms of quality on a set of popular publicly available datasets. The library has a GPU implementation of learning algorithm and a CPU implementation of scoring algorithm, which are significantly faster than other gradient boosting libraries on ensembles of similar sizes.

Subjects: **Machine Learning (cs.LG)**; Mathematical Software (cs.MS); Machine Learning (stat.ML)
Cite as: arXiv:1810.11363 **[cs.LG]**
(or arXiv:1810.11363v1 **[cs.LG]** for this version)

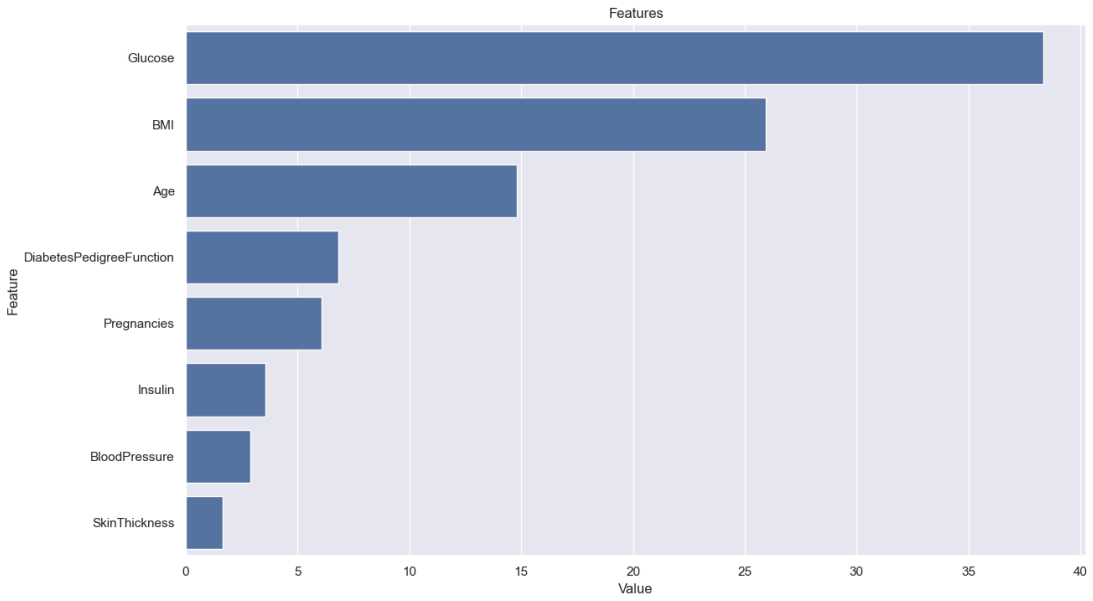
Submission history
From: Anna Veronika Dorogush [view email]
[v1] Wed, 24 Oct 2018 13:08:24 UTC (862 KB)

```
#####################################################
# CatBoost
#####################################################

catboost_model = CatBoostClassifier(random_state=17, verbose=False)

cv_results = cross_validate(catboost_model, X, y, cv=5, scoring=["accurac
y", "f1", "roc_auc"])

cv_results['test_accuracy'].mean() #np.float64(0.7735251676428148)
cv_results['test_f1'].mean() # np.float64(0.6502723851348231)
cv_results['test_roc_auc'].mean() #np.float64(0.8378923829489867)

#Hyperparameter optimization
catboost_params = {"iterations": [200, 500],
                   "learning_rate": [0.01, 0.1],
                   "depth": [3, 6]}


catboost_best_grid = GridSearchCV(catboost_model, catboost_params, cv
=5, n_jobs=-1, verbose=True).fit(X, y)
```

```
catboost_final = catboost_model.set_params(**catboost_best_grid.best_params_, random_state=17).fit(X, y)

cv_results = cross_validate(catboost_final, X, y, cv=5, scoring=["accuracy", "f1", "roc_auc"])

cv_results['test_accuracy'].mean() #np.float64(0.7721755368814192)
cv_results['test_f1'].mean() #np.float64(0.6322580676028953)
cv_results['test_roc_auc'].mean() #np.float64(0.842001397624039)
```

Feature Importance:

```
################################################
# Feature Importance
################################################

def plot_importance(model, features, num=len(X), save=False):
    feature_imp = pd.DataFrame({'Value': model.feature_importances_, 'Feature': features.columns})
    plt.figure(figsize=(10, 10))
    sns.set(font_scale=1)
    sns.barplot(x="Value", y="Feature", data=feature_imp.sort_values(by="Value",
                                                     ascending=False)[0:num])
    plt.title('Features')
    plt.tight_layout()
    plt.show()
    if save:
        plt.savefig('importances.png')

plot_importance(rf_final, X)
plot_importance(gbm_final, X)
plot_importance(xgboost_final, X)
plot_importance(lgbm_final, X)
plot_importance(catboost_final, X)
```



RandomSearchCV

- **RandomizedSearchCV** ise, belirli bir hiperparametre uzayından (gridinden) **rastgele olarak örneklenmiş** belirli sayıda kombinasyonu test eden bir optimizasyon tekniğidir. Tüm olası kombinasyonları denemek yerine, sizin belirlediğiniz sayıda (`n_iter`) kombinasyonu rastgele seçer ve sadece bu kombinasyonları değerlendirir.

Özellik	**GridSearchCV**	**RandomizedSearchCV**
Arama Stratejisi	Belirtilen her bir hiperparametrenin tüm olası değer kombinasyonlarını sistematik olarak dener.	Belirtilen hiperparametre dağılımından rastgele seçilen belirli sayıda kombinasyonu dener.
Kapsamlılık	Arama uzayındaki en iyi kombinasyonu garanti eder (eğer tüm ilgili değerler gridde ise).	En iyi kombinasyonu kaçırma ihtimali vardır, ancak genellikle iyi bir kombinasyon bulur.
Hesaplama Maliyeti	Çok yüksek olabilir, özellikle çok sayıda hiperparametre veya geniş değer aralıkları varsa.	Belirlenen n_iter sayısına bağlıdır, genellikle daha düşüktür.
Kullanım Alanı	Küçük hiperparametre uzayları ve/veya çok önemli olan belirli bir aralıktaki ince ayarlar için idealdir.	Büyük hiperparametre uzayları ve/veya hiperparametrelerin en iyi değerleri hakkında az ön bilgiye sahip olunduğunda daha verimlidir.

Örnek Seçimi	Deterministik (her zaman aynı sonuçları verir).	Rastgeledir (aynı kodu farklı çalıştırmalarda farklı sonuçlar verebilir, random_state ile kontrol edilebilir).
Bilgi Kaybı	Yok (tüm grid noktalarını inceler).	Potansiyel olarak en iyi kombinasyonu kaçırabilir.

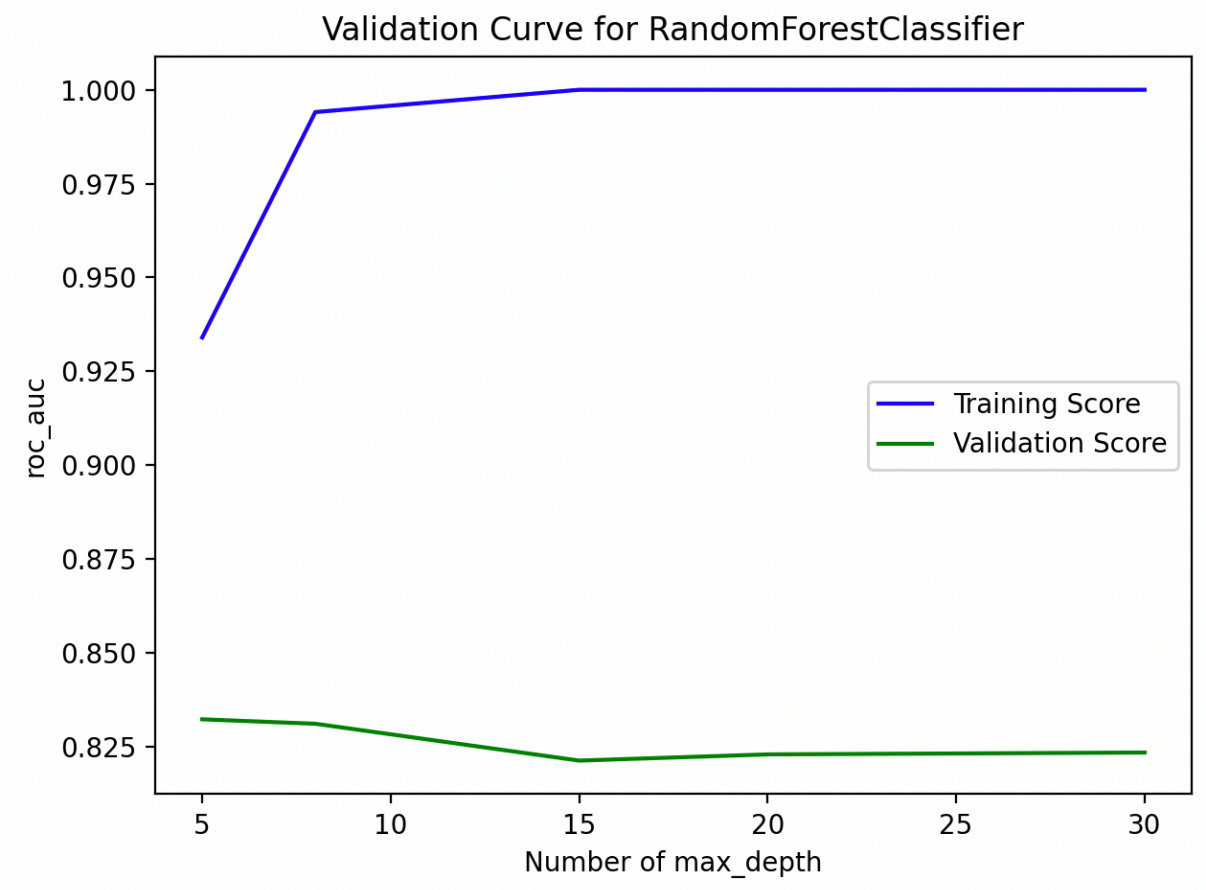
```
#################################
# Hyperparameter Optimization with RandomSearchCV (BONUS)
#################################

rf_model = RandomForestClassifier(random_state=17)

rf_random_params = {"max_depth": np.random.randint(5, 50, 10),
                    "max_features": [3, 5, 7, "auto", "sqrt"],
                    "min_samples_split": np.random.randint(2, 50, 20),
                    "n_estimators": [int(x) for x in np.linspace(start=200, stop=150
0, num=10)]}

rf_random = RandomizedSearchCV(estimator=rf_model,
                               param_distributions=rf_random_params,
                               n_iter=100,  # denenecek parametre sayısı
                               cv=3,
                               verbose=True,
                               random_state=42,
                               n_jobs=1) ## ← reduce parallel workers

rf_random.fit(X, y)


rf_random.best_params_
#{'n_estimators': 344, 'min_samples_split': np.int64(37), 'max_features': 3,
'max_depth': np.int64(37)}

rf_random_final = rf_model.set_params(**rf_random.best_params_, random
_state=17).fit(X, y)

cv_results = cross_validate(rf_random_final, X, y, cv=5, scoring=["accurac
```

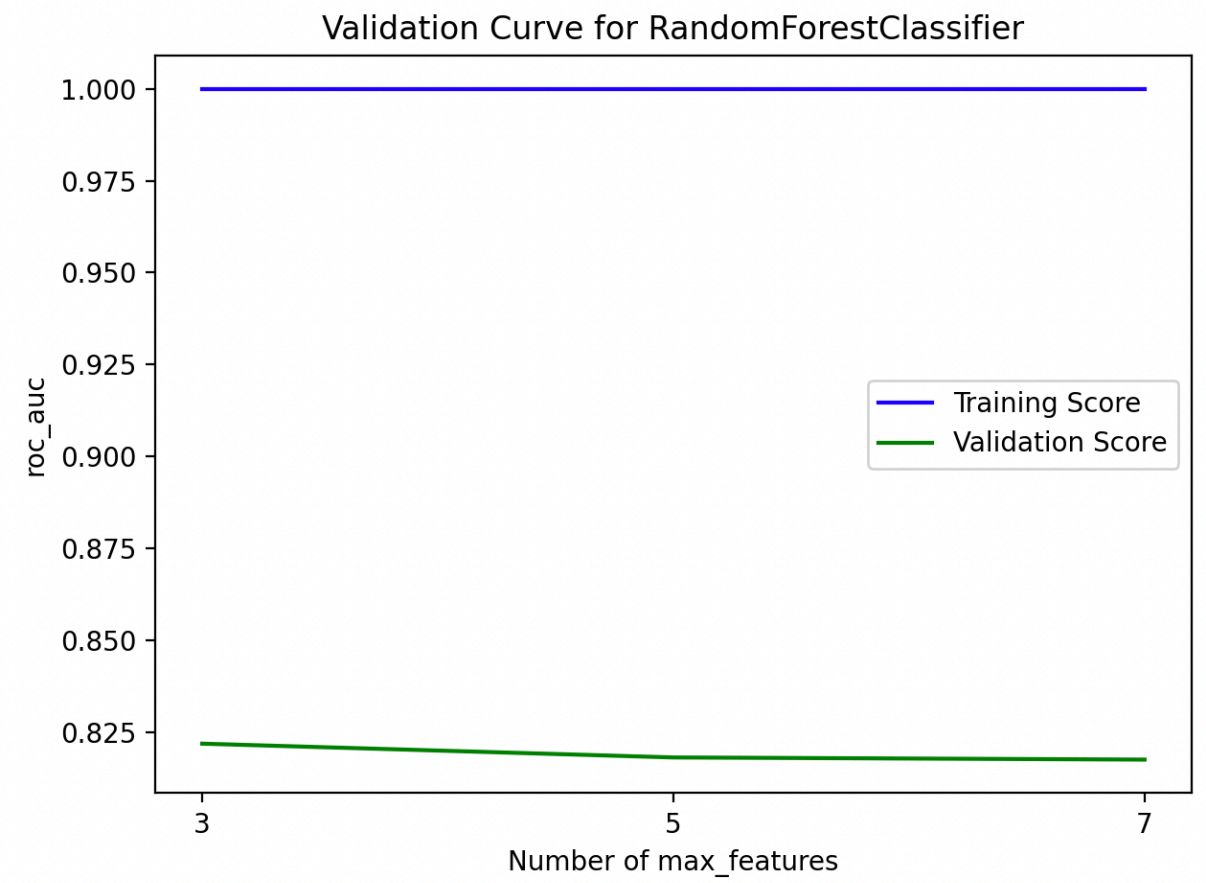
```
y", "f1", "roc_auc"])

cv_results['test_accuracy'].mean() #np.float64(0.7656905186316951)
cv_results['test_f1'].mean() #np.float64(0.6288480975957893)
cv_results['test_roc_auc'].mean() #np.float64(0.8372997903563941)
```

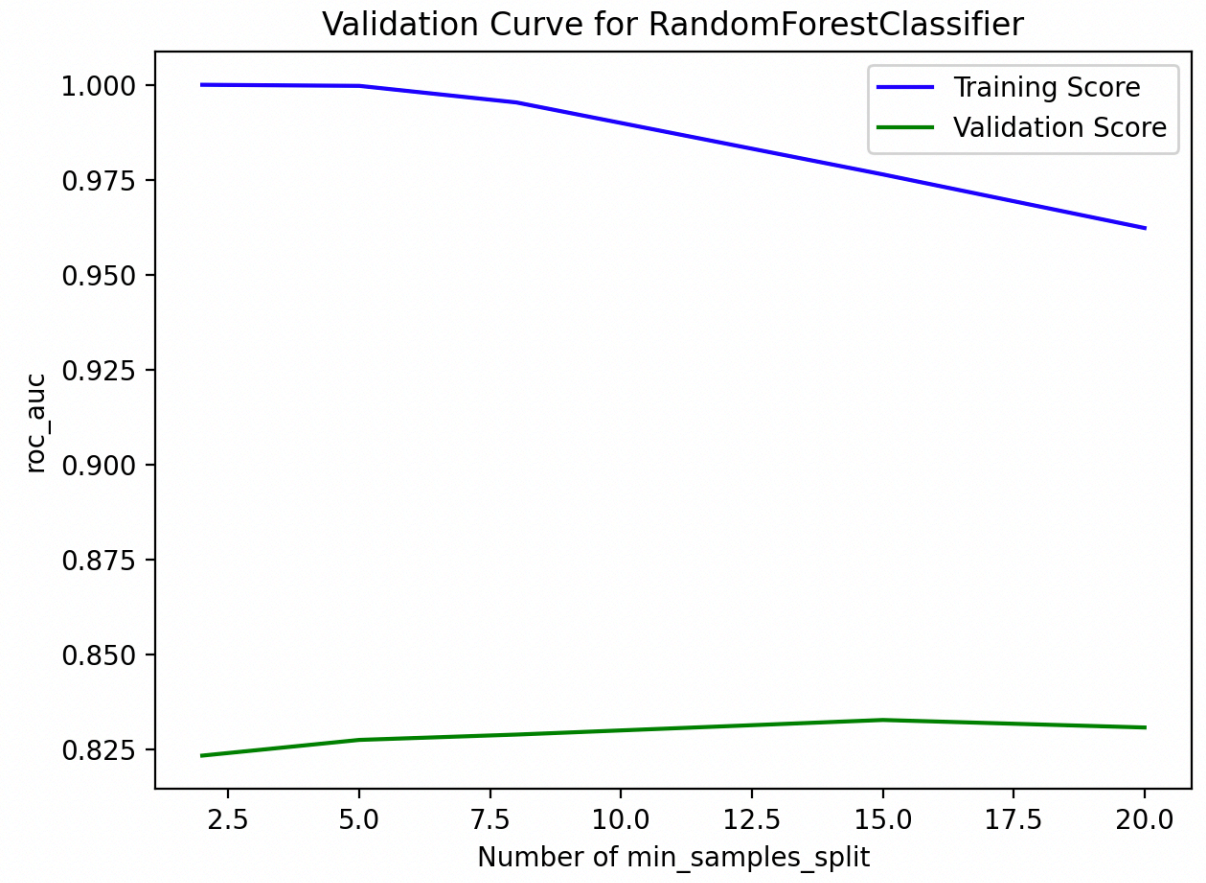
Öğrenme Eğrileriyle Model Karmaşıklığını İnceleme

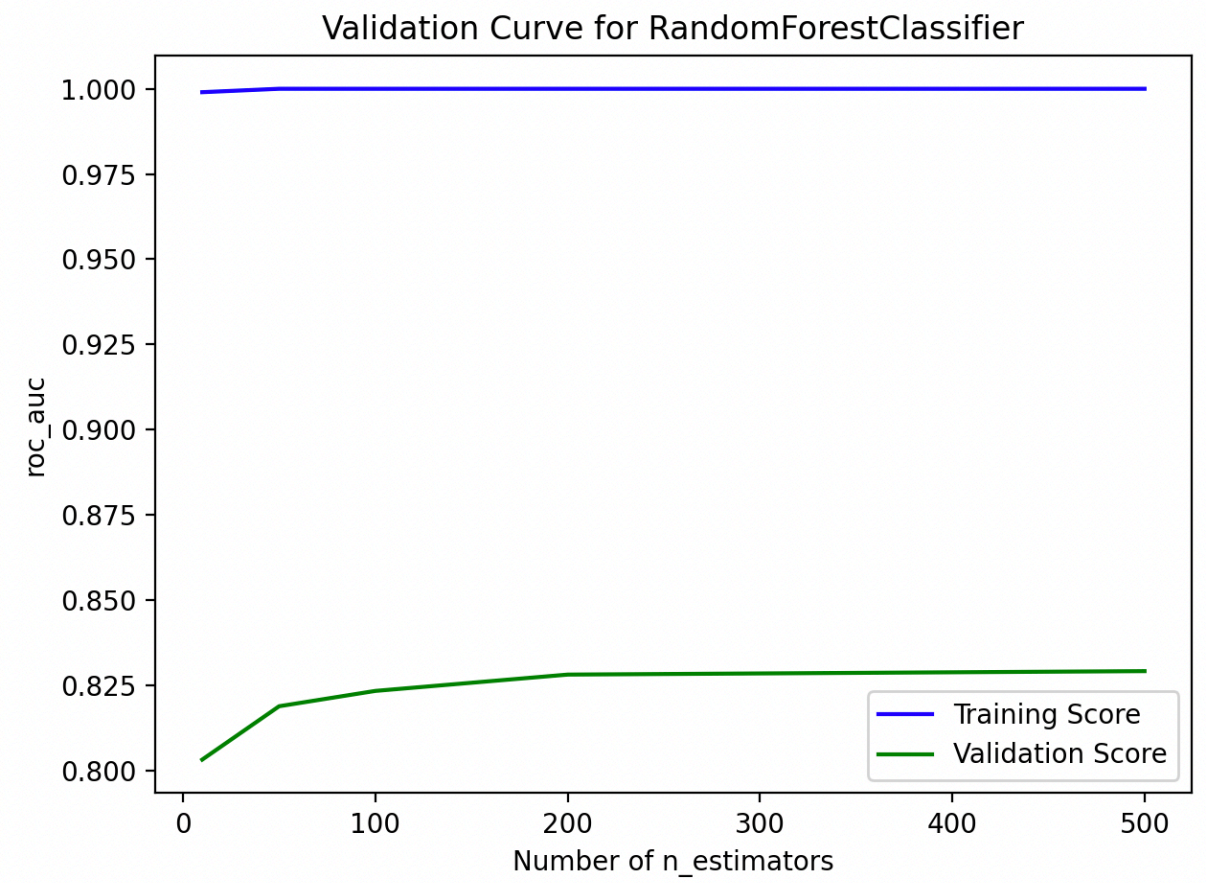
```
##################################
# Analyzing Model Complexity with Learning Curves (BONUS)
##################################

def val_curve_params(model, X, y, param_name, param_range, scoring="ro
c_auc", cv=10):
    train_score, test_score = validation_curve(
        model, X=X, y=y, param_name=param_name, param_range=param_ra
nge, scoring=scoring, cv=cv)

    mean_train_score = np.mean(train_score, axis=1)
    mean_test_score = np.mean(test_score, axis=1)

    plt.plot(param_range, mean_train_score,
             label="Training Score", color='b')

    plt.plot(param_range, mean_test_score,
             label="Validation Score", color='g')

    plt.title(f"Validation Curve for {type(model).__name__}")
    plt.xlabel(f"Number of {param_name}")
    plt.ylabel(f"{scoring}")
    plt.tight_layout()
    plt.legend(loc='best')
    plt.show(block=True)
```

```
rf_val_params = [["max_depth", [5, 8, 15, 20, 30, None]],
                 ["max_features", [3, 5, 7, "auto"]],
                 ["min_samples_split", [2, 5, 8, 15, 20]],
                 ["n_estimators", [10, 50, 100, 200, 500]]]


rf_model = RandomForestClassifier(random_state=17)

for i in range(len(rf_val_params)):
    val_curve_params(rf_model, X, y, rf_val_params[i][0], rf_val_params[i][1])

rf_val_params[0][1] #[5, 8, 15, 20, 30, None]
```




Validation Curve for RandomForestClassifier
roc_auc
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
Training Score
Validation Score
3
5
7
Number of max_features


Validation Curve for RandomForestClassifier
roc_auc
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
Training Score
Validation Score
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Number of min_samples_split


Validation Curve for RandomForestClassifier
roc_auc
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0
100
200
300
400
500
Number of n_estimators
Training Score
Validation Score