

İÇİNDEKİLER

	Sayfa
İçindekiler	1
Şekiller Listesi	2
BÖLÜM BİR – GİRİŞ	4
BÖLÜM İKİ – ALGORİTMA TEORİLERİ	5
2.1 Lojistik Regresyon	5
2.2 Destek Vektör Makine Sınıflayıcıları (SVC)	5
2.3 Karar Ağaçları	6
2.4 Rassal Ormanlar	7
2.5 K – En Yakın Komşuluk (K-NN)	8
2.6 AdaBoost	9
2.7 Naive Bayes	10
2.8 XGBoost	11
2.9 Yapay Sinir Ağları	11
BÖLÜM ÜÇ – UYGULAMA	13
3.1 Uygulama 1	13
3.2 Uygulama 2	21
BÖLÜM DÖRT – SONUÇ	24
BÖLÜM BEŞ – KAYNAKÇA	25

ŞEKİLLER LİSTESİ

	Sayfa
Şekil 2.2.1 SVC Çekirdek Fonksiyonları	5
Şekil 2.3.1 Karar Ağacı Örneği	6
Şekil 2.4.1 Rassal Ormanlar Örneği	7
Şekil 2.5.1 K-NN Örneği	8
Şekil 2.6.1 AdaBoost Ağırlıkları	9
Şekil 2.6.2 AdaBoost Izgara Yapısı	9
Şekil 2.6.3 AdaBoost Sınıflandırma Izgarası	9
Şekil 2.7.1 Mail Histogram	10
Şekil 2.7.2 Koşullu Olasılıklar	10
Şekil 2.9.1 Nöron Çalışma Prensibi	12
Şekil 2.9.2 Örnek Ağ Modeli	12
Şekil 2.9.3 Geri Yayılm Sistemi	12
Şekil 3.1.1 Uygulama 1 Veri Seti	13
Şekil 3.1.2 Kayıp Gözlem Kontrolü	14
Şekil 3.1.3 Aykırı Gözlem Kontrolü	14
Şekil 3.1.4 Veri Setini Ayırma	14
Şekil 3.1.5 Lojistik Regresyon Modeli	15
Şekil 3.1.6 Lojistik Regresyon Metrikleri	15
Şekil 3.1.7 Lojistik Regresyon Karmaşıklık Matrisi	15
Şekil 3.1.8 SVC Modeli	16
Şekil 3.1.9 SVC Metrikleri	16
Şekil 3.1.10 XGBoost Modeli	16
Şekil 3.1.11 XGBoost Metrikleri	16
Şekil 3.1.12 Karar Ağacı Modeli	17
Şekil 3.1.13 Karar Ağacı Metrikleri	17
Şekil 3.1.14 Naive Bayes Modeli	17

	Sayfa
Şekil 3.1.15 Naive Bayes Metrikleri	17
Şekil 3.1.16 Yapay Sinir Ağları Modeli	18
Şekil 3.1.17 Yapay Sinir Ağları Metrikleri	18
Şekil 3.1.18 K-NN Modeli	18
Şekil 3.1.19 K-NN Metrikleri	18
Şekil 3.1.20 Rassal Ormanlar Modeli	19
Şekil 3.1.21 Rassal Ormanlar Metrikleri	19
Şekil 3.1.22 Modellerin Kıyaslaması	19
Şekil 3.1.23 Optimize Edilecek Rassal Orman Parametreleri	20
Şekil 3.1.24 Optimize Edilmiş Rassal Ormanlar Modeli	20
Şekil 3.1.25 Optimize Edilmiş Rassal Ormanlar Metrikleri	20
Şekil 3.2.1 Uygulama 2 Veri Seti	21
Şekil 3.2.2 Modellerin Kurulum Kodu	22
Şekil 3.2.3 Tüm Modellerin Kıyaslaması	22
Şekil 3.2.4 Optimize Edilmiş XGBoost Modeli	23
Şekil 3.2.5 Optimize Edilmiş XGBoost Metrikleri	23
Şekil 3.2.6 Optimize Edilmiş XGBoost Karmaşıklık Matrisi	23

BÖLÜM BİR

GİRİŞ

Günümüzde istatistik ve matematiğin mucize çocuğu olarak adlandırılabilceğimiz makine öğrenmesi algoritmalarının arasında bile sınıflandırma algoritmaları gerek kullanım alanları gerekse insanların hayatlarının içine girmiş olmasından kaynaklı olarak her zaman kendine ait bir yeri olacaktır. Günümüzde sınıflandırma algoritmaların en çok kullanıldığı alanlar tıp, müşteri tabanlı hizmet sektörü, elektronik öneri sistemleri, spam algılama sistemleri gibi alanlardır.

Netflix, Spotify gibi büyük markalar tarafından kullanılan kişisel öneri sistemlerinin altında yatan mantık basitçe sınıflandırma algoritmasıdır. Örnek vermek gerekirse; X şarkısını dinlediğinizi ve sevdığınızı düşünelim bu sizi beğendi yani 1 olarak algoritmaya girdi olarak atar. Çıktı olarak ise X şarkısını beğenmiş diğer kişilerin beğendi şarkıları size beğenebileceğiniz şarkılar adı altında sunar. Buradan sonrası tamamen otomatik bir öğrenme sürecini tetikler her beğendiniz ve beğenmediğiniz şarkı için yeniden sınıflandırılır ve buna göre öneri alırsınız.

Tıp sektöründe ise genellikle hastalıkların sınıflandırılmasında kullanılan sınıflandırma algoritmalarının birçok kişinin hayatına ciddi ölçüde etki ettiğini söylemek yanlış olmaz. Sınıflandırma algoritmaları sayesinde erken tanı mantığı inanılmaz bir hız kazanmakla beraber gün geçtikçe verilerin artması ile birlikte tahmin doğruluğu artıyor. Artan tahmin doğruluğu doktorların, hastalara uygulayacağı tedaviyi seçmesinde ve başlamasında yardımcı oluyor. Özette hastalar insanların karar verme yetisinin üstüne bir makine tarafından hasta olup olmadığını erkenden öğrenip, aksiyona geçebiliyor.

BÖLÜM İKİ

METOT

Raporun bu bölümünde proje genelinde kullanılan ve yararlanılan sınıflama algoritmalarının teorilerinden detaylı bir şekilde bahsedilecektir.

2.1 Lojistik Regresyon

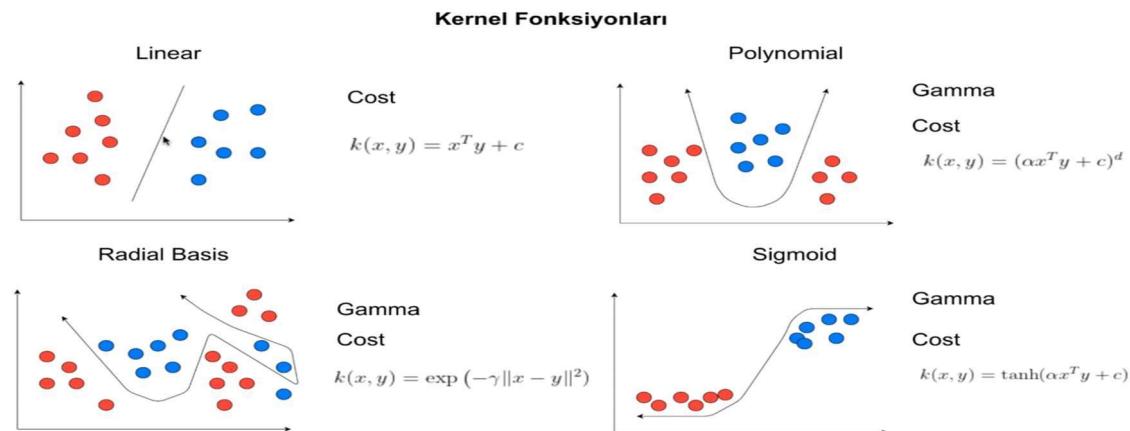
Genellikle ikili sınıflandırma için kullanılan, doğrusal model oluşturulduktan sonra çıkan sonuçları sigmoid fonksiyonundan geçirerek olasılık değerleri üretmesi mantığına dayanır. Üretilen olasılık değerleri, algoritma tarafından belirlenen (0.5) veya kullanıcın belirlediği eşik değerine göre eşik değerinin altında kalanlar 0 üzerinde kalanlar 1 şeklinde sınıflandırmaya yarayan bir algoritma. Özellikle doğrusal ilişkinin olduğu iki sınıfı veri setlerini sınıflandırmasında başarılı olan bu model çok sınıfı veri setlerinde de çalışmakta olup, sınıflandırmayı (sınıf sayısı – 1) olacak sayıda farklı eşik değeri ile yapar.

2.2 Destek Vektör Makine Sınıflayıcıları (SVC)

Sınıflandırma yapabilmek için ve sınıfları birbirinden ayırabilmek için bir hiper düzlem çizmesi gereken SVC algoritması başta iki sınıfı problemleri çözmek için geliştirilmiş olmakla birlikte günümüzde çok sınıfı problemler içinde oldukça kullanışlıdır.

Algoritma kendi içerisinde bazı gözlemleri destek noktası olarak alır ve bu destek noktalara olan uzaklığı en az yapacak şekilde bir hiper düzlem çizerek sınıfları ayırtmayı amaçlar. Adından da anlaşılacağı üzere destek noktaları hiper düzlem ve ayırtma için kilit rol oynamaktadır.

Hiper düzlemler düz bir doğru olabileceği gibi radyal bazlı da olabilir.



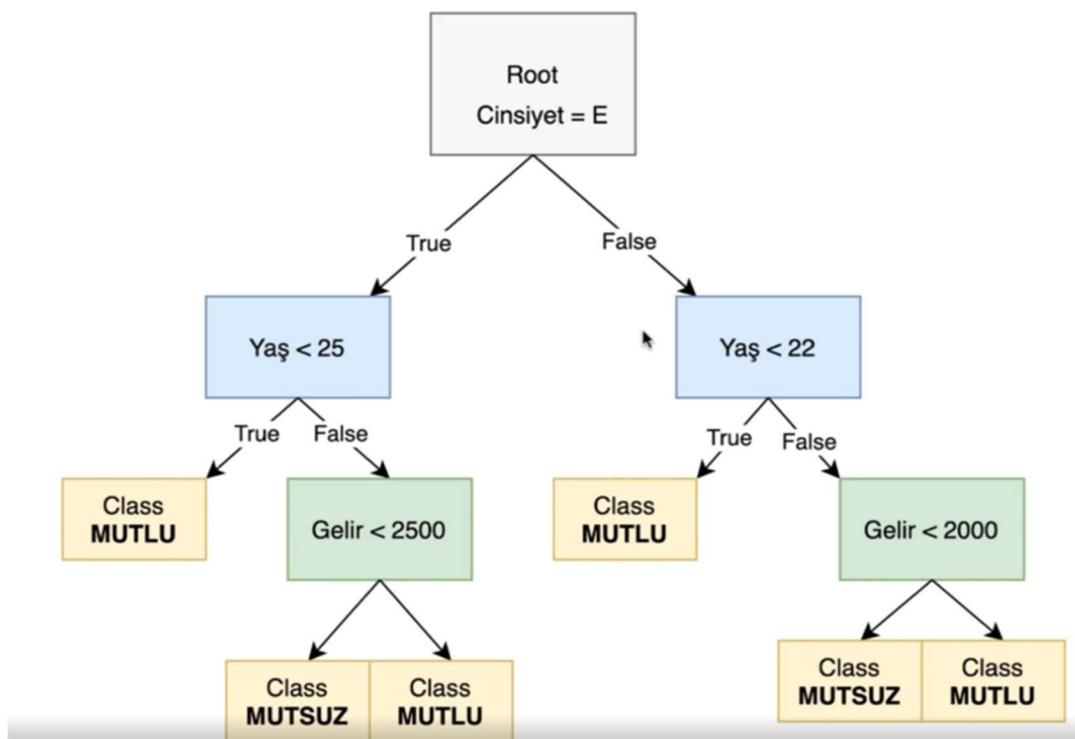
Şekil 2.2. 1 SVC Çekirdek Fonksiyonları

2.3 Karar Ağaçları

Doğrusal olmayan ve gözetimli öğrenme algoritmalarının başını çeken karar ağaçları basitçe bir insanın karar mantığını taklit eder.

Kök ve terminal noktalarının önemleri ve algoritmanın nereden karar vermeye ve neye göre karar vereceği ‘Bilgi Kazancı’ yöntemi ile tespit edilir. Örnek vermek gerekirse;

Şekil 2.3.1 de gözüktüğü üzere yaş değişkeni maaş değişkeninden bir seviye yukarıda bulunmaktadır bu yaş değişkenin gelire göre daha iyi bir öğrenici olduğunu gösterir.



Şekil 2.3. 1 Karar Ağacı Örneği

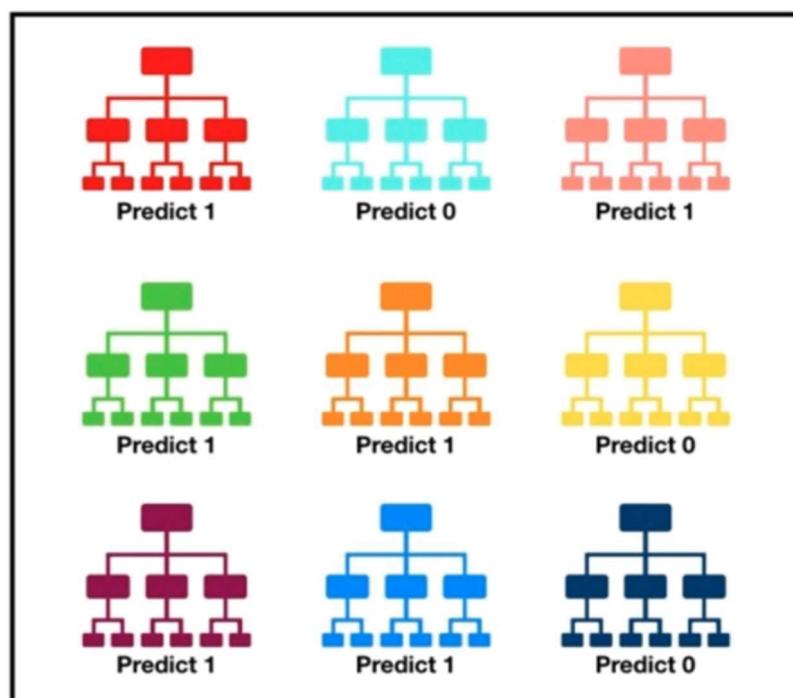
2.4 Rassal Ormanlar

Karar ağaçları algoritmasını ve torbalama tekniği bir arada kullanan bu gözetimli öğrenme yöntemi en basit haliyle belirtilen sayıda üretilmiş karar ağaçlarının olduğu bir orman olarak düşünülebilir.

Algoritma tamamen rasgele çalışmaktadır yani elimizde dokuz adet bağımsız değişkeni bulunan bir veri setimiz olduğunu hayal edelim. Algoritma eğer karar ağaçlarını oluştururken kullanması gereken en fazla bağımsız değişken sayısını üç olarak belirtir ve dokuz adet ağaç üretmesini söyleyerek.

Algoritma toplam dokuz adet olan bağımsız değişkenlerimizden her defasında rasgele üç adet seçip karar ağaçları oluşturacaktır ve bu işlemi belirttiğimiz sayıda ağaç üretmek için tekrarlayacaktır. Sonucunda oluşan ormandan aldığı çıktılar ile çoğunluk olan sınıfı göre karar verme işlemini gerçekleştirir.

Örnek vermek gerekirse; dokuz karar ağaçından oluşan rassal ormanımızda altı adet karar ağaçımız tahmin değerini ‘1’, üç adet karar ağaçımız ise tahmin değerini ‘0’ olarak tahmin etmiş olsun. Bu durumda rassal ormanlar algoritması çoğunlukta olan sayıya göre sınıflandırma yapar ve o gözlemi ‘1’ olarak sınıflar.



Tally: Six 1s and Three 0s
Prediction: 1

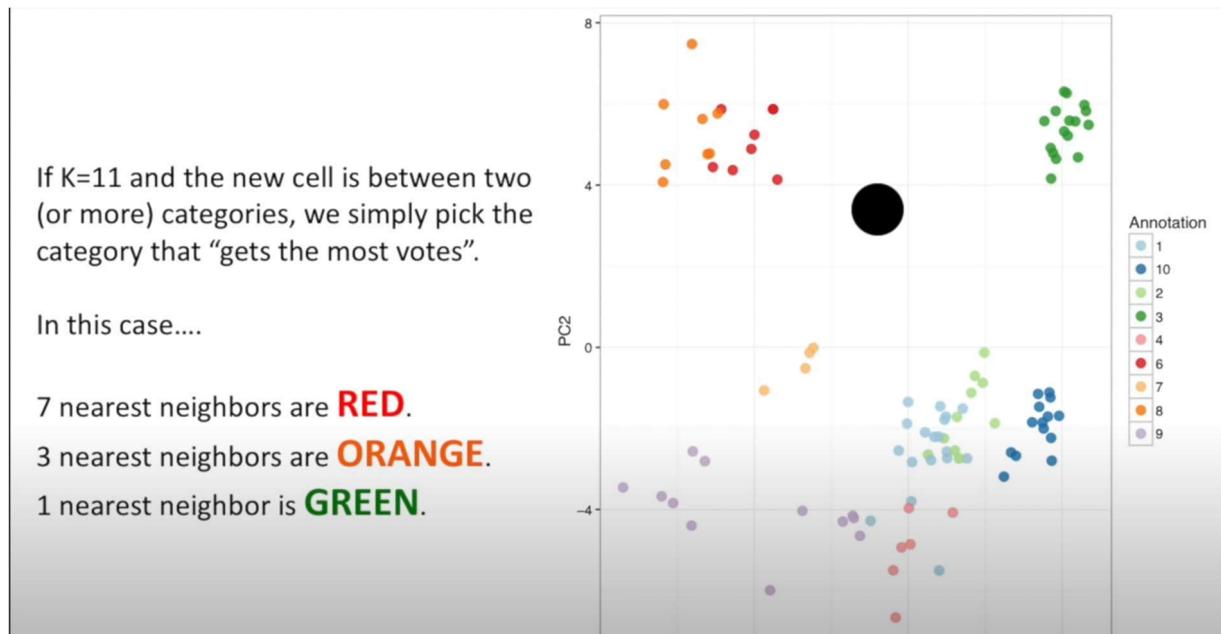
Şekil 2.4. 1 Rassal Ormanlar Örneği

2. 5 K – En Yakın Komşuluk (K-NN)

K- En yakın komşuluk algoritması, eğitim verisi sonrası gruplanan verilerin, yeni bir gözlem geldiğinde gözleme en yakın K adet gözlem incelenmesi ve bu inceleme sonucu üstünlük kuran grup olarak sınıflanmasını sağlar.

Şekil 2.5.1 de gözüktüğü üzere yeni gelen siyah noktaya uzak on bir(K) noktanın grup sayıları hesaplanıyor. Hesaplama sonrasında on bir gözlemden yedi adedi kırmızı, üç adedi turuncu ve bir adedi yeşil olarak sınıflanmış durumda. Yeni gelen siyah noktamıza en yakın on bir gözlemden yedi adedi kırmızı olduğu için algoritma bu yeni noktanın kırmızı olacağını tahmin eder.

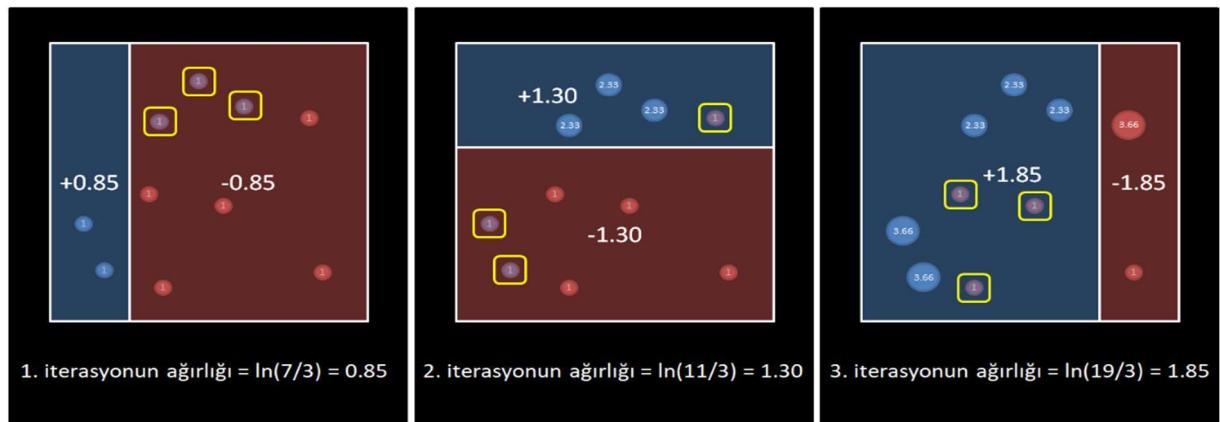
Bu on bir nokta belirlenirken yeni gözleme olan uzaklıklarını Öklid ya da benzeri uzaklık hesapları ile hesaplanır ve minimum olan on bir gözlem en yakın komşu olarak algoritmaca etiketlenir.



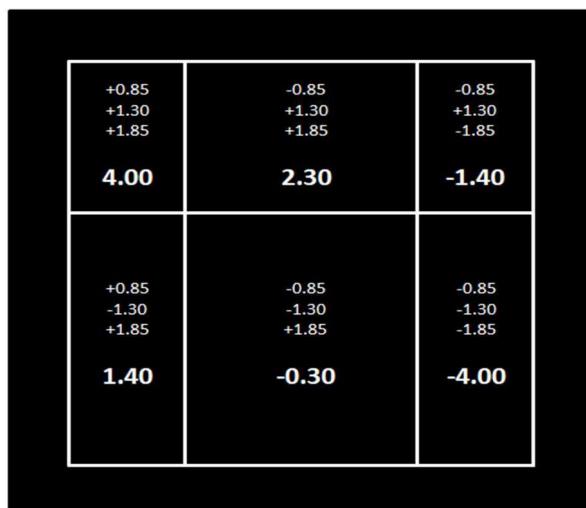
Şekil 2.5. 1 K-NN Örneği

2.6 AdaBoost

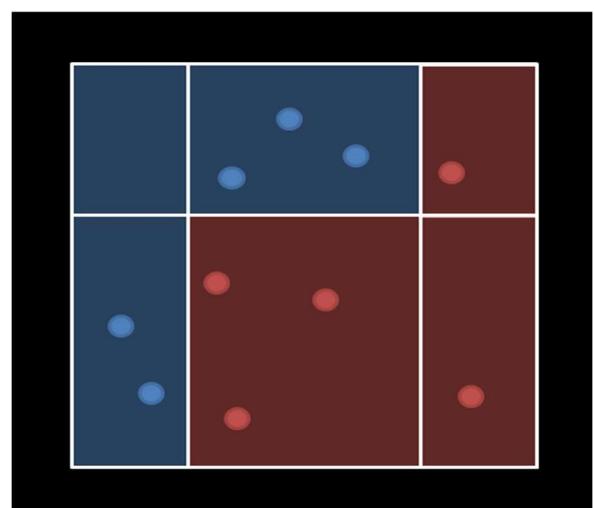
Adaptif Yükseltme veya AdaBoost, en basit yükseltme algoritmalarından biridir. Genellikle, karar ağaçları modelleme için kullanılır. Her biri son modeldeki hataları düzeltten çoklu sıralı modeller oluşturular. AdaBoost, yanlış tahmin edilen gözlemlere ağırlık atar ve sonraki model bu değerleri doğru şekilde tahmin etmek için çalışır.



Şekil 2.6. 1 AdaBoost Ağırlıkları



Şekil 2.6. 2 AdaBoost Izgara Yapısı



Şekil 2.6. 3 AdaBoost Sınıflandırma Izgarası

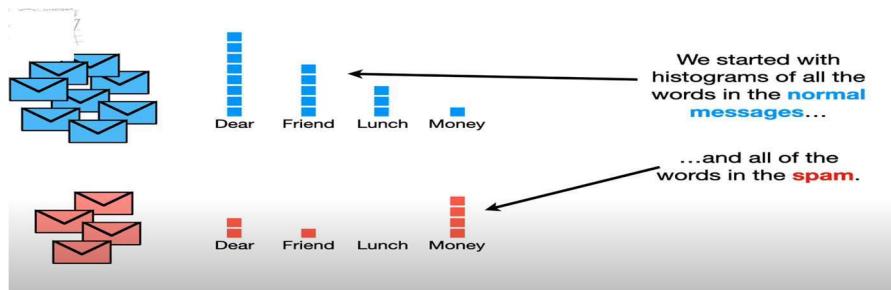
Şekil 2.6.1 de gözüktüğü üzere AdaBoost algoritması her yinelemeye bir önceki yanlış tahmin ettiği gözlemlerin örneklem içerisindeki ağırlıklarını arttırarak, onları doğru tahmin etmeyeceklerdir. Her yinelemeye sonucu ayrılan bölgelere ağırlık katsayısı atanır ve katsayılar Şekil 2.6.2 de görüldüğü üzere toplanarak bir sonuca varılır, varılan sonucun pozitif veya negatif olmasına bağlı olarak oluşan izgara yapısı artık veriyi sınıflamak için hazırlıdır.

Izgara yapısında pozitif değer alan alanlar Mavi, negatif değer alan alanlar ise Kırmızı olarak sınıflanmıştır.

2.7 Naive Bayes

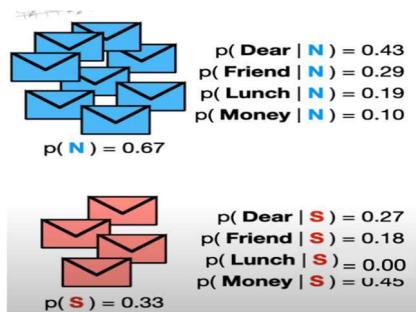
Naive Bayes algoritması kelimenin tam anlamı ile Bayes'in koşullu olasılık teorisi üzerine kurulmuş bir sınıflandırma algoritmasıdır. Algoritma her bir sınıf ve durum için koşullu olasılık hesaplayıp, yeni gelen gözlem için daha önceden hesapladığı koşullu olasılık durumlarının çarpılması sonucu ham bir olasılık değeri elde eder.

Farklı sınıflar ve durumlar altında koşullu olasılığı hesaplanan yeni gözlemimizin temelde sınıf sayımız kadar farklı olasılık değerleri olur. Elde edilen olasılık değerlerinden en yüksek olanı hangi sınıfa ait ise gözlemede o sınıfa ait olacaktır.



Şekil 2.7 1 Mail Histogram

Elimizde sekiz adet normal ve dört adet spam olan e-posta olduğunu düşünelim. Bu maillerde geçen kelimelerin histogramı çizildiğinde;



Şekil 2.7 2 Koşullu Olasılıklar

Yandaki koşullu olasılıklar ve olasılık bulunur. Bu bağlamda diyelim ki size “Dear Friend” diye bir e-posta geldi. Bu spam mıdır? Yoksa normal midir?

Normal Olması : $P(N) * P(\text{Dear} | N) * P(\text{Friend} | N) = 0,083$

Spam Olması : $P(S) * P(\text{Dear} | S) * P(\text{Friend} | S) = 0,016$

Olasılıklara bakıldığından Normal Olması olasılığı daha yüksek olduğu için algoritma “dear friend” içeren bir maili normal olarak sınıflayacaktır.

Fakat naive bayes algoritması olasılık temelli çalıştığından, “Lunch Money Money Money Money” diye bir e-posta gelirse spam maillerde lunch kelimesi geçmediği için Lunch içeren bir mailinin spam olma olasılığı sıfır olacaktır. Bu bağlamda bunun önüne geçmek için alpha olarak adlandırılan bir sayı kadar her sınıfı ekleme yapılır. Örnek vermek gerekirse;

Spam ve normal maillerdeki her kelimenin frekansını bir arttırarak sıfır olması sorununu ortadan kaldırabilir.

$$\text{Bu durumda } P(\text{Lunch Money}^4) = P(S) * P(\text{Lunch} | S) * P(\text{Money} | S)^4$$

$$0.33 * 0.09 * 0.45^4 = 0.00122 \quad \text{Ve böylelikle spam olarak algılanır}$$

2.8 XGBoost

XGBoost yani tam adı ile Extreme Gradient Boost(Aşırı Gradyan Yüksetlme) algoritması, gradyan yükseltme makineleri algoritmasının geliştirilmiş bir versiyonu olarak düşünülebilir. Gradyan yükseltme algoritmasının yanı sıra XGBoost algoritması bir başlangıç tahmin değeri ile yola başlar, bu değer 0,5 olarak ön tanımlanmıştır.

Daha sonra veri setindeki gözlemler için artıkları üretip artıklardan yola çıkarak karar ağaçları oluşturmaya başlıyor. Rassal ormanlardan farklı olarak, oluşturulan m^{nci} ağaç ($m-1$)'inci ağaç'a bağlı olarak oluşturuluyor.

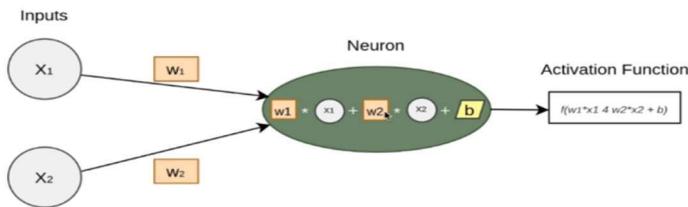
İlk ağaç oluşturulduğunda yapraklardaki terminal değerlerinin belirlenmesi için ‘Benzerlik Skoru’ denen seçilen değerin veri setini ne kadar iyi ayırip ayırmadığını gösteren bir skora bakılıyor. Bu skorun büyük olması seçilen eşik değerinin veri setini diğerlerine kıyasla çok daha iyi ayırtıldığı gösteriyor.

XGBoost algoritması çok fazla karmaşık verilerde daha iyi çalışmasının yanı sıra aşırı eğitilme durumuna yatkın bir algoritma, bu durumun önüne geçmek için ağaçlarda budamaya gidilmesi gibi bir konsept ortaya atılmış durumda. Bu konsept budama parametresi yani gammanın (γ) , kazanç (sağ yaprak + sol yaprak – kök benzerliği) değerinden farkının alınması ile oluşan değer inceleniyor. Eğer kazanç, gamma farkı pozitif ise dalın kalacağı, eğer negatif ise dalın budanacağı anlamına geliyor. Yapılan bu işlem eğitim verisi üzerindeki aşırı öğrenme durumunu yok etmek amaçlı uygulanıyor.

2.9 Yapay Sinir Ağları (YSA)

Yapay sinir ağları, insan beyninin bilgi işleme tekniğinden esinlenerek geliştirilmiş bir bilgi işlem teknolojisidir. YSA ile basit biyolojik sinir sisteminin çalışma şekli taklit edilir. Yani biyolojik nöron hücrelerinin ve bu hücrelerin birbirleri ile arasında kurduğu sinaptik bağın dijital olarak modellenmesidir. Nöronlar çeşitli şekillerde birbirlerine bağlanarak ağlar oluştururlar. Bu ağlar öğrenme, hafızaya alma ve veriler arasındaki ilişkiyi ortaya çıkarma kapasitesine sahiptirler. Diğer bir ifadeyle, YSA'lar, normalde bir insanın düşünme ve gözlemlemeye yönelik doğal yeteneklerini gerektiren problemlere çözüm üretmektedir. Bir insanın, düşünme ve gözlemleme yeteneklerini gerektiren problemlere yönelik çözümler üretebilmesinin temel sebebi ise insan beyninin ve dolayısıyla insanın sahip olduğu yaşayarak veya deneyerek öğrenme yeteneğidir.

Neural Network - Nöronun Görevi



Şekil 2.9. 1 Nöron Çalışma Prensibi

Nöronlar basitçe girdileri daha önceden hesaplanmış ağırlıklar ile çarpıp nöronun kendi yanlışlık değerini ekleyip aktivasyon fonksiyonuna ya da çıktı olarak gönderir.

Şekil 2.9.2 incelediğinde örnek bir ağ modelinin temel çalışma mantığı anlaşılmaktadır.

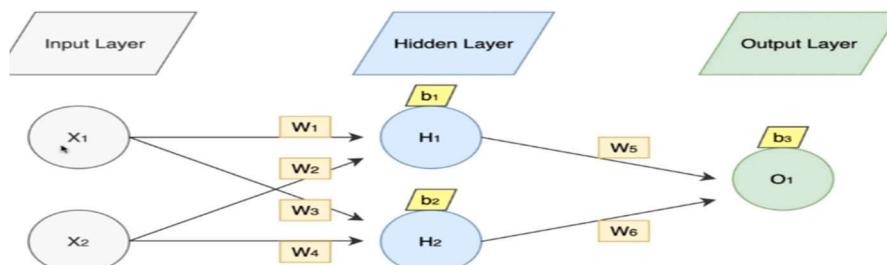
$$H1 \text{ Nöronunun sonucu: } x1 \cdot w1 + x2 \cdot w2 + b1 = H1$$

$$H2 \text{ Nöronunun sonucu: } w3 \cdot x1 + x2 \cdot w4 + b2 = H2$$

Bu durumda;

$$O1 \text{ çıktısının sonucu: } w5 \cdot H1 + w6 \cdot H2 + b3$$

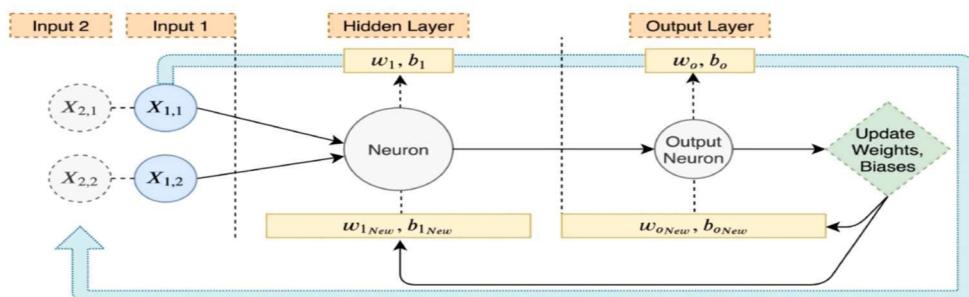
Örnek Bir Ağ Modeli



Şekil 2.9. 2 Örnek Ağ Modeli

Modelin eğitiminde temel bir yeri olan geri yayılım sistemi modelin ileri besleme sonucu oluşturduğu çıktılar ile eğitim setindeki gerçek değerleri karşılaştırarak, yeni ağırlık, yanlışlık değerlerini ve öğrenim oranını optimize eder. Bu işlem eğitim verisi ve tahmin verileri arasındaki hata en düşük oluncaya dek devam eder.

Backpropagation



Şekil 2.9. 3 Geri Yayılım Sistemi

BÖLÜM ÜÇ

UYGULAMA

Raporun bu bölümünde, metot kısmında görmüş olduğumuz sınıflandırma algoritmalarını iki farklı veri seti üzerinde deneyip sonuçları kıyaslayacağız.

3.1 Uygulama 1

Seeds Veri Seti Tanıtımı

Bağımsız değişkenler:

1. Alan
2. Çevre
3. Yoğunluk
4. Çekirdek Uzunluğu
5. Çekirdek Genişliği
6. Asimetri Katsayısı
7. Çekirdek Oluğu Uzunluğu

Bağımlı değişken:

1. Tohum Tipi: (0-1-2) Çok Sınıflı

	area	perimeter	compactness	lengthOfKernel	widthOfKernel	asymmetryCoefficient	lengthOfKernelGroove	seedType
0	15.26	14.84	0.8710	5.763	3.312	2.221	5.220	1
1	14.88	14.57	0.8811	5.554	3.333	1.018	4.956	1
2	14.29	14.09	0.9050	5.291	3.337	2.699	4.825	1
3	13.84	13.94	0.8955	5.324	3.379	2.259	4.805	1
4	16.14	14.99	0.9034	5.658	3.562	1.355	5.175	1
...
205	12.19	13.20	0.8783	5.137	2.981	3.631	4.870	3
206	11.23	12.88	0.8511	5.140	2.795	4.325	5.003	3
207	13.20	13.66	0.8883	5.236	3.232	8.315	5.056	3
208	11.84	13.21	0.8521	5.175	2.836	3.598	5.044	3
209	12.30	13.34	0.8684	5.243	2.974	5.637	5.063	3

210 rows × 8 columns

Şekil 3.1. 1 Uygulama 1 Veri Seti

210 Satır ve yedisi bağımsız, biri bağımlı değişken olmak üzere sekiz sütundan oluşan veri setimizde amaç bağımlı değişkenimiz olan tohum tipini, yedi adet bağımlı değişkeni kullanarak doğru tahmin edebilmek.

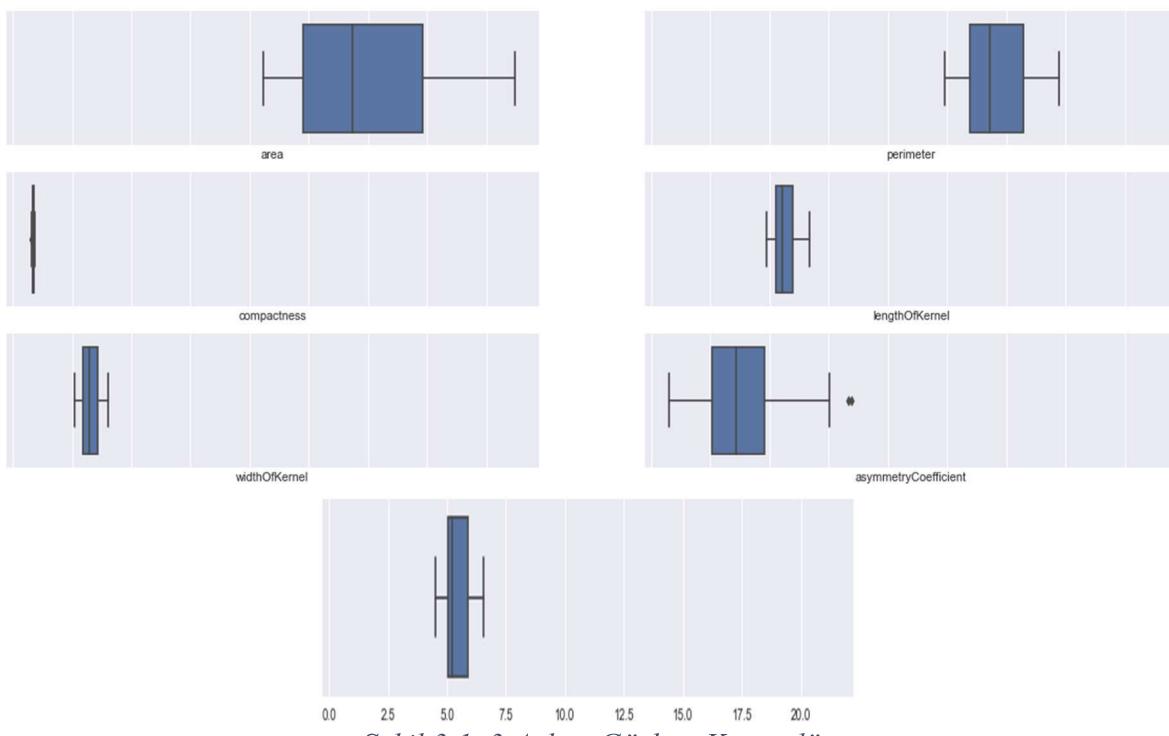
İlk olarak veri setimizde eksik gözlem olup olmadığını kontrol etmek amaçlı kodumuzu çalıştırıp, Şekil 3.1.2'de de görüldüğü üzere verimizin herhangi bir sütununda eksik gözlem olmadığını anlıyoruz.

```
seeds.isna().value_counts()

area    perimeter   compactness  lengthOfKernel  widthOfKernel  asymmetryCoefficient  lengthOfKernelGroove  seedType
False      False       False        False         False          False                False           False            210
dtype: int64
```

Şekil 3.1. 2 Kayıp Gözlem Kontrolü

Verimizde herhangi bir aykırı değer olup olmadığını gözlemlemek amacıyla kutu grafikleri oluşturulmuştur.



Şekil 3.1. 3 Aykırı Gözlem Kontrolü

Şekil 3.1.3'te de görüldüğü üzere yalnızca “asymmetryCoefficient” sütununda aykırı değerler gözlenmiş olup IQR yöntemi ile aykırı değerler veri setinden atılmıştır.

```
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=42,stratify=y,test_size=0.25)
```

Şekil 3.1. 4 Veri Setini Ayırma

Şekil 3.1.4'te görüldüğü üzere hazırlanmış olan verimizi test ve eğitim verisi olmak üzere iki parçaya bölyoruz. Buradaki “stratify” değeri ham verinin bağımlı değişkenindeki farklı sınıf oranlarını korumak üzerine bağımlı değişken üzerinden ayarlanır.

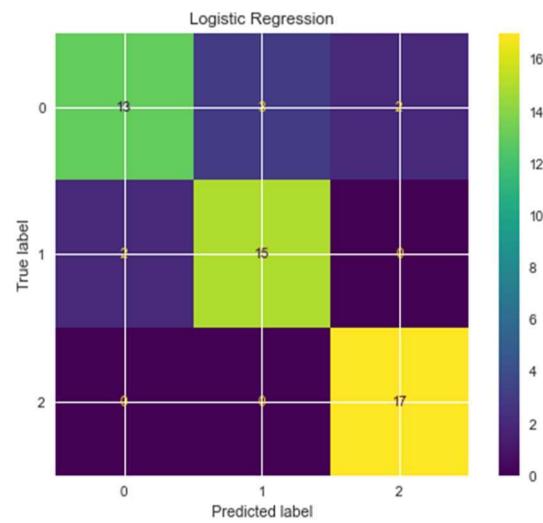
Şekil 3.1.5'te de görüldüğü üzere ilk olarak verimiz için ‘Saga’ çözümü ve ‘Ridge’ düzeltme katsayı ile Lojistik Regresyon modeli kuruyoruz.

```
logisticModel = LogisticRegression("l2",solver="saga",dual=False).fit(X_train,y_train)
predLogistic = logisticModel.predict(X_test)
acsLogi = accuracy_score(y_test,predLogistic)
print("\t\tLogistic Regression Results\n")
print(classification_report(y_test,predLogistic))
plot_confusion_matrix(logisticModel,X_test,y_test);
plt.title("Logistic Regression");
aucLogi = roc_auc_score(y_test, logisticModel.predict_proba(X_test), multi_class='ovr')
```

Şekil 3.1. 5 Lojistik Regresyon Modeli

Logistic Regression Results				
	precision	recall	f1-score	support
0	0.87	0.72	0.79	18
1	0.83	0.88	0.86	17
2	0.89	1.00	0.94	17
accuracy			0.87	52
macro avg	0.86	0.87	0.86	52
weighted avg	0.86	0.87	0.86	52

Şekil 3.1. 6 Lojistik Regresyon Metrikleri



Şekil 3.1. 7 Lojistik Regresyon Karmaşıklık Matrisi

Şekil 3.1.7'de bulunan Karmaşıklık Matrisinden hesaplanıldığı gibi Şekil 3.1.6'da da hesaplanmış kesinlik ('Precision') ve hafıza ('Recall') metriklerini görebiliriz. Burada modelin doğruluğu ('Accuracy') ise 0,87 olarak bulunmuş ve yeterlidir.

Şekil 3.1.8'de de görüldüğü üzere ikinci olarak verimiz içi çekirdeği 'Rbf' (Radyal Hiper Düzlem) olan SVC modeli kuruyoruz.

```
modelSVC = SVC(kernel="rbf",probability=True).fit(X_train,y_train)
predModelSVC = modelSVC.predict(X_test)
acsSVC = accuracy_score(y_test,predModelSVC)
print(classification_report(y_test,predModelSVC))
plot_confusion_matrix(modelSVC,X_test,y_test)
plt.title("Support Vector Classifier");
aucSVC = roc_auc_score(y_test, modelSVC.predict_proba(X_test), multi_class='ovr')
```

Şekil 3.1. 8 SVC Modeli

	precision	recall	f1-score	support
0	0.87	0.72	0.79	18
1	0.88	0.88	0.88	17
2	0.85	1.00	0.92	17
accuracy			0.87	52
macro avg	0.87	0.87	0.86	52
weighted avg	0.87	0.87	0.86	52

Şekil 3.1. SVC Metrikleri

Şekil 3.1.9'da da görüldüğü üzere modelimizin doğruluk oranı 0,87 olmakla birlikte diğer metriklerimiz neredeyse Lojistik Regresyon modelimizin metrikleri ile aynı.

Üçüncü olarak kurdugumuz XGBoost modelimizde öğrenme derecesini 0,01 olarak alarak modelimizi kurduk.

```
modelXGB = XGBClassifier(use_label_encoder=False,
                         learning_rate=0.01,
                         gamma=1,
                         eval_metric="mlogloss").fit(X_train,y_train)
predModelXGB = modelXGB.predict(X_test)
aucXGB = roc_auc_score(y_test, modelXGB.predict_proba(X_test), multi_class='ovr')
plt.title("XGBoost Confusion Matrix")
print(classification_report(y_test,predModelXGB))
```

Şekil 3.1. 10 XGBoost Modeli

	precision	recall	f1-score	support
0	0.87	0.72	0.79	18
1	0.94	0.88	0.91	17
2	0.81	1.00	0.89	17
accuracy			0.87	52
macro avg	0.87	0.87	0.86	52
weighted avg	0.87	0.87	0.86	52

Şekil 3.1. 11 XGBoost Metrikleri

Şekil 3.1.11'deki metrikler incelendiğinde kurulan ilk iki model ile arasında büyük bir fark olmadığı gözlenmektedir.

Dördüncü olarak verimiz için bir de Karar Ağacı Sınıflandırıcısı ile model kuruyoruz.

```
modelDTC = DecisionTreeClassifier(random_state=42).fit(X_train,y_train)
predModelDTC = modelDTC.predict(X_test)
acsDTC = accuracy_score(y_test,predModelDTC)
aucDTC = roc_auc_score(y_test, modelDTC.predict_proba(X_test), multi_class='ovr')
plt.title("Decision Tree Confusion Matrix")
print(classification_report(y_test,predModelDTC))
```

Şekil 3.1. 12 Karar Ağacı Modeli

	precision	recall	f1-score	support
0	0.94	0.83	0.88	18
1	0.94	0.94	0.94	17
2	0.89	1.00	0.94	17
accuracy			0.92	52
macro avg	0.92	0.92	0.92	52
weighted avg	0.92	0.92	0.92	52

Şekil 3.1. 13 Karar Ağacı Metrikleri

Şekil 3.1.13'te bulunan metrikler incelendiğinde Karar Ağacı modelimizin diğer üç modelden daha performansı çalıştığını söyleyebiliriz lakin henüz tüm modelleri kıyaslamadığımız için en iyisi diyemeyiz.

Özellikle metin verilerinde çok başarılı olan Naive Bayes modelimizi, üstün bir performans bekłentisi olmadan kendi verimiz için kuruyoruz.

```
modelNB = GaussianNB().fit(X_train,y_train)
predModelNaive = modelNaive.predict(X_test)
acsNB = accuracy_score(y_test,predModelNaive)
aucNB = roc_auc_score(y_test, modelNB.predict_proba(X_test), multi_class='ovr')
plt.title("Gaussian Naive Bayes Confusion Matrix")
print(classification_report(y_test,predModelNaive))
```

Şekil 3.1. 14 Naive Bayes Modeli

	precision	recall	f1-score	support
0	0.82	0.78	0.80	18
1	0.94	0.88	0.91	17
2	0.84	0.94	0.89	17
accuracy			0.87	52
macro avg	0.87	0.87	0.87	52
weighted avg	0.87	0.87	0.86	52

Şekil 3.1. 15 Naive Bayes Metrikleri

Fakat Şekil 3.1.15'te görüldüğü üzere Naive Bayes modeli, verimiz için neredeyse ilk üç model kadar başarılı.

Çok fazla veriye ihtiyaç duyan Yapay Sinir Ağları modelini, kendi veri setimiz için deniyoruz.

```
modelNeu = MLPClassifier().fit(X_train,y_train)
predModelNeu = modelNeu.predict(X_test)
acsNeu = accuracy_score(y_test,predModelNeu)
aucNeu = roc_auc_score(y_test, modelNeu.predict_proba(X_test), multi_class='ovr')
plot_confusion_matrix(modelNeu,X_test,y_test)
plt.title("K-NN Confusion Matrix")
print(classification_report(y_test,predModelNeu))
```

Şekil 3.1. 16 Yapay Sinir Ağları Modeli

	precision	recall	f1-score	support
0	0.82	0.78	0.80	18
1	0.83	0.88	0.86	17
2	0.94	0.94	0.94	17
accuracy			0.87	52
macro avg	0.87	0.87	0.87	52
weighted avg	0.87	0.87	0.86	52

Şekil 3.1. 17 Yapay Sinir Ağları Metrikleri

Fakat Şekil 3.1.17'de görüldüğü üzere veriye olan açlığı ile bilinen Yapay Sinir Ağları algoritması Karar Ağacı modelimiz dışında kurulan diğer tüm modellerle aynı performansı yakalamış gibi görünüyor.

Yedinci olarak K = 5 parametresi ile verimiz için K – En Yakın Komşuluk algoritmasını deniyoruz.

```
modelNN = KNeighborsClassifier(n_neighbors=5).fit(X_train,y_train)
predModelNN = modelNN.predict(X_test)
acsNN = accuracy_score(y_test,predModelNN)
aucNN = roc_auc_score(y_test, modelNN.predict_proba(X_test), multi_class='ovr')
plt.title("K-NN Confusion Matrix")
print(classification_report(y_test,predModelNN))
```

Şekil 3.1. 18 K-NN Modeli

	precision	recall	f1-score	support
0	0.86	0.67	0.75	18
1	0.83	0.88	0.86	17
2	0.85	1.00	0.92	17
accuracy			0.85	52
macro avg	0.85	0.85	0.84	52
weighted avg	0.85	0.85	0.84	52

Şekil 3.1. 19 K-NN Metrikleri

Şekil 3.1.19'de de görüldüğü üzere modelden elde ettiğimiz metrikler ve doğruluk değeri incelendiğinde şu ana kadar kurulan modeller arasında en kötü modelin K - En Yakın Komşuluk modeli olduğunu söyleyebiliriz.

Son olarak Karar Ağacı algoritmasının başarısından sonra, Karar Ağaçlarından oluşan Rassal Ormanlar algoritmasını veri setimiz üzerinde uyguluyoruz.

```
modelRF = RandomForestClassifier(n_estimators=300).fit(X_train,y_train)
predModelRF = modelRF.predict(X_test)
acsRF = accuracy_score(y_test,predModelRF)
aucRF = roc_auc_score(y_test, modelRF.predict_proba(X_test), multi_class='ovr')
plt.title("Random Forest Confusion Matrix")
print(classification_report(y_test,predModelRF))
```

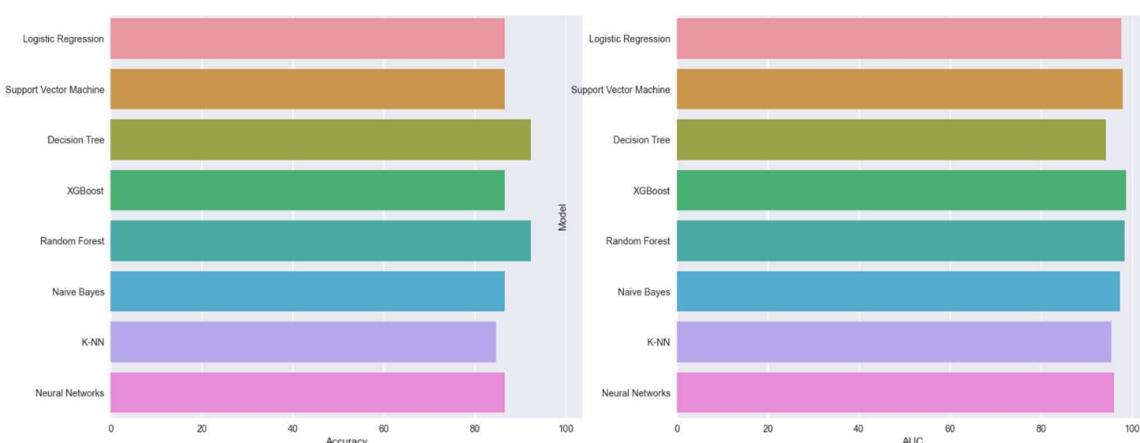
Şekil 3.1. 20 Rassal Ormanlar Modeli

	precision	recall	f1-score	support
0	1.00	0.78	0.88	18
1	0.94	1.00	0.97	17
2	0.85	1.00	0.92	17
accuracy			0.92	52
macro avg	0.93	0.93	0.92	52
weighted avg	0.93	0.92	0.92	52

Şekil 3.1. 21 Rassal Ormanlar Metrikleri

Şekil 3.1.21'de de görüldüğü üzere Karar Ağaçları üzerinden karar veren Rassal Ormanlar modelimiz hemen hemen Karar Ağacı modeli ile aynı performansa sahip.

Şekil 3.1.22'de görüldüğü üzere kurmuş olduğumuz tüm modeller için doğruluk ve 'Roc-Auc' skorlarını hesaplayıp kıyasladık. Kıyaslama sonuçlarına bakıldığında her iki metrikte de en yüksek değerleri elde ederek, en iyi başarıyı yakalayan modelimiz ise Rassal Ormanlar modeli olmuştur.



Şekil 3.1. 22 Modellerin Kıyaslansması

Daha önce yaptığımız model kıyaslaması sonucu en başarılı model olarak seçtiğimiz Rassal Ormanlar modelini farklı hiper parametreler ile optimize etmek için GridSearchCV fonksiyonundan yararlanıyoruz.

```
param_grid = {"n_estimators":np.arange(1,500,50),
              "criterion":["gini","entropy"],
              "max_depth":np.arange(1,20),
              "min_samples_split":[1,2,3,4],
              "max_features":["auto","sqrt","log2"],
              "bootstrap": [True, False]}
clf = GridSearchCV(RandomForestClassifier(), param_grid, n_jobs=-1, verbose=1)
clf.fit(X_train,y_train)
```

Şekil 3.1. 23 Optimize Edilecek Rassal Orman Parametreleri

‘GridSearchCV’ fonksiyonu ile optimize edilmiş hiper parametreler ile modelimizi tekrar kuruyoruz.

```
model = RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                               criterion='entropy', max_depth=18, max_features='auto',
                               max_leaf_nodes=None, max_samples=None,
                               min_impurity_decrease=0.0, min_impurity_split=None,
                               min_samples_leaf=1, min_samples_split=3,
                               min_weight_fraction_leaf=0.0, n_estimators=401,
                               n_jobs=None, oob_score=False, random_state=None,
                               verbose=0, warm_start=False);
model.fit(X_train,y_train)
```

Şekil 3.1. 24 Optimize Edilmiş Rassal Ormanlar Modeli

	precision	recall	f1-score	support
0	1.00	0.78	0.88	18
1	0.94	1.00	0.97	17
2	0.85	1.00	0.92	17
accuracy			0.92	52
macro avg	0.93	0.93	0.92	52
weighted avg	0.93	0.92	0.92	52

Şekil 3.1. 25 Optimize Edilmiş Rassal Ormanlar Metrikleri

Şekil 3.1.25'te de görüldüğü üzere modelimiz %92 doğruluk oranına sahip olmakla birlikte %98,6'lık bir ‘AUC’ skoruna da sahiptir.

3.2 Uygulama 2

Adults Veri Seti Tanıtımı:

Bağımsız değişkenler:

1. Yaş
2. Çalışma Sınıfı
3. Fnlwgt
4. Eğitim
5. Eğitim Numarası
6. Medeni Hal
7. Meslek
8. İlişki
9. Irk
10. Cinsiyet
11. Sermaye Kazancı
12. Sermaye Kaybı
13. Haftalık Çalışma Saati
14. Uyruk

Bağımlı değişken:

1. Yıllık Gelir (≤ 50 : 0, > 50 : 1)

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	target
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	$\leq 50K$
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	$\leq 50K$
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	$\leq 50K$
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	$\leq 50K$
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	$\leq 50K$
...
45217	33	Private	245211	Bachelors	13	Never-married	Prof-specialty	Own-child	White	Male	0	0	40	United-States	$\leq 50K$
45218	39	Private	215419	Bachelors	13	Divorced	Prof-specialty	Not-in-family	White	Female	0	0	36	United-States	$\leq 50K$
45219	38	Private	374983	Bachelors	13	Married-civ-spouse	Prof-specialty	Husband	White	Male	0	0	50	United-States	$\leq 50K$
45220	44	Private	83891	Bachelors	13	Divorced	Adm-clerical	Own-child	Asian-Pac-Islander	Male	5455	0	40	United-States	$\leq 50K$
45221	35	Self-emp-inc	182148	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	60	United-States	$> 50K$

45222 rows × 15 columns

Şekil 3.2. 1 Uygulama 2 Veri Seti

45522 Satır ve on dördü bağımsız, biri bağımlı değişken olmak üzere toplam on beş sütundan oluşan veri setimizde amaç bağımlı değişkenimiz olan yıllık geliri, on dört adet bağımlı değişkeni kullanarak doğru tahmin edebilmek.

XGBoost, Rassal Ormanlar, Karar Ağaçları ve Lojistik Regresyon gibi algoritmalar model kurulurken bağımlı değişkenlerin önem düzeylerini hesapladığı için bu modelleri kurarken kullanılacak olan bağımsız değişkenleri modelden seçtirdik, lakin aynı durum K-NN, SVC, YSA ve Naive Bayes algoritmalarında bulunmadığından, bu algoritmalarla kullanılacak olan bağımsız değişkenler ‘chi²’ önem testine göre seçilmiştir.

Şekil 3.1.2’de de görüleceği üzere tüm modellerin aynı anda kurulması için kod hazırlanmış olup kod çıktı olarak kurulan modelin ismi ve hesaplanmış metrikleri göstermektedir.

```

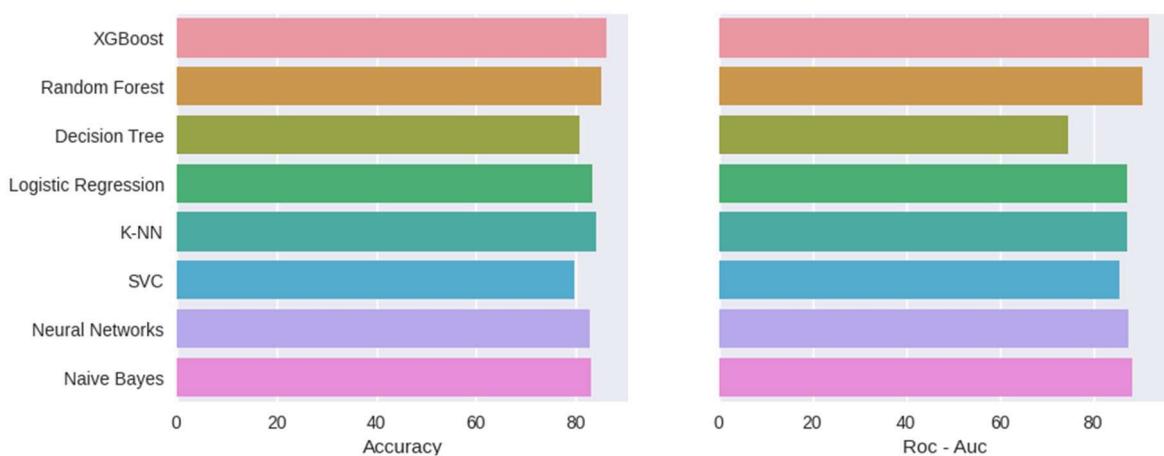
modeller = [XGBClassifier(),RandomForestClassifier(),DecisionTreeClassifier(),LogisticRegression()]
modeller2 = [KNeighborsClassifier(),SVC(),MLPClassifier(),GaussianNB()]
modelnames = ["XGBoost","Random Forest","Decision Tree","Logistic Regression"]
modelnames2 = ["K-NN","SVC","Neural Networks","Naive Bayes"]
pd.DataFrame({})

def modeldensec(modeller):
    global scores
    skb = SelectKBest(chi2).fit(X_train,y_train)
    X_trainchi = skb.transform(X_train)
    for i,j in zip(modeller,range(len(modelnames))):
        clf = SelectFromModel(i).fit(X_train,y_train)
        Xtrain = clf.transform(X_train)
        testscores = cross_validate(Xtrain,y_train,scoring=["accuracy","roc_auc","f1","recall","precision"],cv=5,n_jobs=-1)
        auc = testscores["test_roc_auc"].mean()
        accu = testscores["test_accuracy"].mean()
        f1 = testscores["test_f1"].mean()
        reca = testscores["test_recall"].mean()
        pre = testscores["test_precision"].mean()
        scores = scores.append({"Model":modelnames[j],"Accuracy":accu,"Roc - Auc":auc,"F1":f1,"Recall":reca,"Precision":pre},ignore_index=True)

    for z,v in zip(modeller2,range(len(modelnames2))):
        testscores = cross_validate(z,Xtrain,y_train,scoring=["accuracy","roc_auc","f1","recall","precision"],cv=5,n_jobs=-1)
        auc = testscores["test_roc_auc"].mean()
        accu = testscores["test_accuracy"].mean()
        f1 = testscores["test_f1"].mean()
        reca = testscores["test_recall"].mean()
        pre = testscores["test_precision"].mean()
        scores = scores.append({"Model":modelnames2[v],"Accuracy":accu,"Roc - Auc":auc,"F1":f1,"Recall":reca,"Precision":pre},ignore_index=True)

```

Şekil 3.2. 2 Modellerin Kurulum Kodu



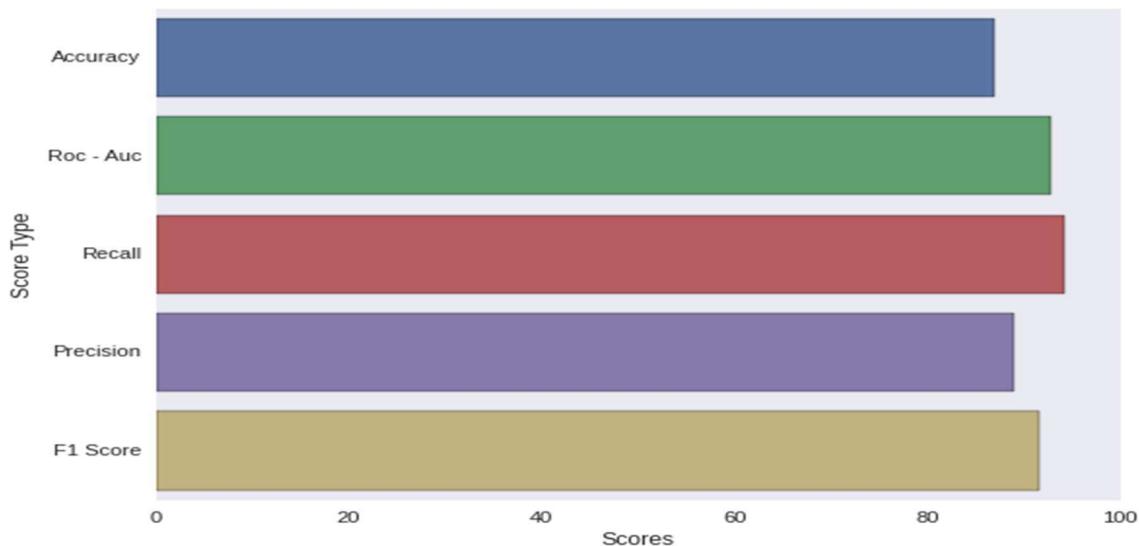
Şekil 3.2. 3 Tüm Modellerin Kıyaslaması

Şekil 3.2.3’e bakıldığından hem doğruluk hem de ‘AUC’ skoru olarak XGBoost modelinin diğer modellere nazaran daha iyi performans verdiği gözükmemektedir. Bu bağlamda XGBoost modelimizin parametreleri optimize edilecektir.

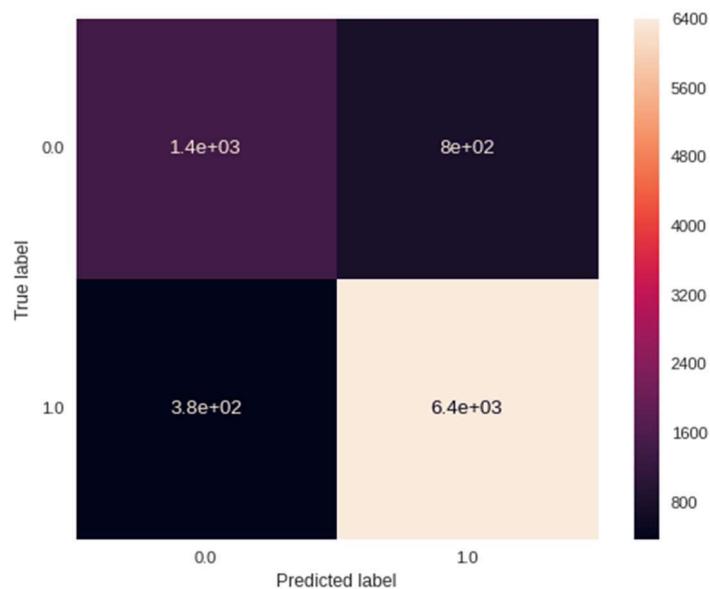
Optimize edilmiş parametreler ile kurulmuş model Şekil 3.2.4'te gözükmemektedir.

```
clf = SelectFromModel(XGBClassifier()).fit(X_train,y_train)
x_train = pd.DataFrame((SelectFromModel(XGBClassifier()).fit(X_train,y_train)).transform(X_train))
x_train.columns = fmap
x_train.rename(columns = lambda x: x.replace(' ', '_'), inplace=True)
x_test = (SelectFromModel(XGBClassifier()).fit(X_train,y_train)).transform(X_test)
model = XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                      colsample_bynode=1, colsample_bytree=0.5, gamma=0,
                      learning_rate=0.05, max_delta_step=0, max_depth=7,
                      min_child_weight=1, missing=None, n_estimators=250, n_jobs=1,
                      nthread=None, objective='binary:logistic', random_state=0,
                      reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                      silent=None, subsample=1, verbosity=1,tree_method="gpu_hist").fit(x_train,y_train)
scores = cross_validate(model,x_train,y_train,scoring=["accuracy","roc_auc","recall","precision","f1"],cv=5)
```

Şekil 3.2. 4 Optimize Edilmiş XGBoost Modeli



Şekil 3.2. 5 Optimize Edilmiş XGBoost Metrikleri



Şekil 3.2. 6 Optimize Edilmiş XGBoost Karmaşıklık Matrisi

BÖLÜM DÖRT

SONUÇ

Sınıflandırmanın günümüzde çok fazla yeri olduğu aşikâr olmakla birlikte kullanım alanlarının da gün geçtikçe arttığı ve yaygınlaşlığı da bir gerçektir.

Birbirinden farklı sınıflandırma algoritmalarının, birbirine benzer sonuçlar verdiği gibi farklı veri setlerinde farklı davranışlar gösterdiğini görmüş olduk, bu bağlamda sınıflandırma algoritmaları ile çalışırken asıl soru “hangi algoritma benim verim için daha uygun olur?” sorusu olmalı.

Örnek vermek gerekirse, yapmış olduğumuz iki adet sınıflandırma örneğinde daha büyük hacme sahip olan Uygulama 2 veri setinde XGBoost modeli daha iyi performans gösterirken, nispeten daha küçük bir hacme sahip olan Uygulama 1 veri setinde ise Rassal Ormanlar algoritması daha iyi bir sonuç göstermiştir.

XGBoost algoritmasının karmaşıklık ile başa çıkma gücü göz önüne alındığında gayet sıradan bir durum olarak nitelendirilebiliriz.

Sonuç olarak kullanılan birçok sınıflandırma algoritması olmakla beraber, birçoğunun kullanım alanları ve kullanılan veri setine göre yarattığı sonuçlar birbirinden farklı olabilir. Bir sınıflandırma probleminde elimizdeki veriyi inceleyip ona uygun olan algoritmayı bulmak gerekmektedir.

BÖLÜM BEŞ

KAYNAKÇA

- 1- <https://machinelearningmastery.com/xgboost-for-imbalanced-classification/>
- 2- <https://towardsdatascience.com/xgboost-fine-tune-and-optimize-your-model-23d996fab663>
- 3- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html#sklearn.metrics.accuracy_score
- 4- <https://scikit-learn.org/stable/modules/generated/sklearn.metrics.auc.html#sklearn.metrics.auc>
- 5- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html#sklearn.model_selection.cross_validate
- 6- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV
- 7- https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectFromModel.html#sklearn.feature_selection.SelectFromModel
- 8- https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.chi2.html#sklearn.feature_selection.chi2
- 9- <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html?highlight=decision%20tree#sklearn.tree.DecisionTreeClassifier>
- 10- <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html?highlight=randomforest#sklearn.ensemble.RandomForestClassifier>
- 11- <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html?highlight=svc#sklearn.svm.SVC>
- 12- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html?highlight=logistic#sklearn.linear_model.LogisticRegression
- 13- https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html?highlight=gaussian#sklearn.naive_bayes.GaussianNB
- 14- https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html?highlight=mlpclassifier#sklearn.neural_network.MLPClassifier
- 15- <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html?highlight=nearest#sklearn.neighbors.NearestNeighbors>
- 16- https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
- 17- https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html
- 18- https://www.youtube.com/playlist?list=PLblh5JKOoLU0irPgs1SnKO6wqVjKU_sQ
- 19- <https://www.youtube.com/playlist?list=PLblh5JKOoLUKAAtDViTvRGFpphEc24M-QH>

- 20- <https://www.youtube.com/playlist?list=PLblh5JKOoLUL3IJ4-yor0HzkqDQ3JmJkc>
- 21- https://www.youtube.com/playlist?list=PLblh5JKOoLUJjeXUvUE0maghNuY2_5fY6
- 22- <https://www.youtube.com/playlist?list=PLblh5JKOoLUIE96dI3U7oxHaCAbZgfhHk>
- 23- <https://www.youtube.com/playlist?list=PLblh5JKOoLUKxzEP5HA2d-Li7IJkHfXSe>
- 24- <https://www.youtube.com/playlist?list=PLblh5JKOoLUIxGDQs4LFFD--41Vzf-ME1>
- 25- <https://archive.ics.uci.edu/ml/datasets/seeds>
- 26- <https://archive.ics.uci.edu/ml/datasets/Adult>