T.C.
İSTANBUL AYDIN ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

MAKİNE ÖĞRENMESİ YÖNTEMLERİYLE OYUN SUNUCU YÜKÜNÜN TAHMİN EDİLMESİ

YÜKSEK LİSANS TEZİ

Çağdaş ÖZER

Bilgisayar Mühendisliği Ana Bilim Dalı

Bilgisayar Mühendisliği Programı

Mart 2020

T.C.
İSTANBUL AYDIN ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

MAKİNE ÖĞRENMESİ YÖNTEMLERİYLE OYUN SUNUCU YÜKÜNÜN TAHMİN EDİLMESİ

YÜKSEK LİSANS TEZİ

Çağdaş ÖZER
(Y1813.010015)

Bilgisayar Mühendisliği Ana Bilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Doç. Dr. Taner Çevik

Mart 2020

YEMİN METNİ

Yüksek Lisans tezi olarak sunduğum “Makine Öğrenmesi Yöntemleriyle Oyun Sunucu Yükünün Tahmin Edilmesi ” adlı çalışmanın, tezin proje safhasından sonuçlanmasına kadar ki bütün süreçlerde bilimsel ahlak ve etik geleneklere aykırı düşecek bir davranışımın olmadığını, tezdeki bütün bilgileri akademik ve etik kurallar içinde elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara kaynak gösterdiğimi ve yararlandığım eserlerin bibliyografyada gösterilenlerden oluştuğunu, bunlara atıf yaparak yararlanmış olduğumu belirtir ve onurumla beyan ederim.

Çağdaş ÖZER

ÖNSÖZ

Bu çalışmayı yapmamı ve yüksek lisansı başlayıp bitirmem için bana her türlü desteği veren çok değerli hayat ortağım, en iyi arkadaşım, eşim Öğr. Gör. Esra Özer'e; varlığıyla ve gülümsemesiyle bana umut olan oğlum Yunus Emre Özer'e; bilgisi, deneyimi ve sahip olduğum bilgiyi daha iyi aktarabilmem için beni her zaman daha yukarı çekmek isteyen ve bunun için elinden geleni yapan, yenilikçi danışmanım Sayın Doç. Dr. Taner Çevik'e; tezimin aşamalarında hatrımı ve olduğum noktayı soran ve sahip olduğum sorunlara her zaman çözüm getirmek isteyen, bana kendi oğlu gibi davranan kıymetli hocam Prof. Dr. Ali Güneş'e, çözüm odaklı fikirleri ve makale desteğiyle bana yardımcı olan değerli ekip arkadaşım Dr. Öğr. Üyesi Ali Hamitoğlu'na, tecrübesini bana aktarmakta hiç çekinmeyen, kullandığım dili kontrol eden ve bu çalışmayı nasıl daha ileriye taşıyabileceğiz diye fikirlerini paylaşan Hocam Prof. Dr. Naim Ajlouni'ye, Dr. Öğr. Üyesi Ahmet Gürhanlı Hocama ve moral olsun fikir olsun bana her şeyiyle destek olan çok sevdiğim arkadaşım Arş. Gör. Mustafa Takaoğlu'na minnet ve teşekkürlerimi sunuyorum.

Mart, 2020 **Çağdaş ÖZER**

İÇİNDEKİLER

KISALTMALAR

ML : Makine Öğrenmesi (Machine Learning)
STLF : Kısa vadeli yük tahmini (Short Term Load Forecast)
MTLF : Orta vadeli yük tahmini (Mid Term Load Forecast)
LTLF : Uzun vadeli yük tahmini (Long Term Load Forecast)
AI : Yapay Zekâ (Artificial Intelligence)
CNN : Konvolüsyonlu Sinir Ağı (Convolutional Neural Network)
TP : Gerçek pozitif (True Positive)
TN : Gerçek negatif (True Negative)
FP : Yanlış pozitif (False Positive)
FN : Yanlış negatif (False Negative)
AUC : Eğrinin altındaki alan (Area Under Curve)
NB : Naif Bayes (Naive Bayes)
GLM : Genelleştirilmiş Doğrusal Model (Generalized Linear Model)
LR : Lojistik Regresyon (Logistic Regression)
FLM : Hızlı Büyük Marj (Fast Large Margin)
DT : Karar Ağaçları (Decision Trees)
GBT : Gradyan Arttırılmış Ağaçlar (Gradient Boosted Trees)
SVM : Destek Vektör Makinesi (Support Vector Machine)
RFE : Özyinelemeli Özellik Ortadan Kaldırılması (Recursive Feature Elimination)
OCR : Optik Karakter Tanıma (Optical Character Recognition)
MMORPG : Devasa çok oyunculu çevrimiçi rol yapma oyunu (Massively Multiplayer Online Role-playing Game)
PCA : Temel Bileşenler Analizi (Principal Component Analysis)
KLT : Karhunen–Loève dönüşümü (Karhunen–Loève Transformation)
MLE : Maksimum Olabilirlik Tahmini (Maximum Likelihood Estimation)
RF : Rastgele Orman (Random Forest)

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

MAKİNE ÖĞRENMESİ YÖNTEMLERİYLE OYUN SUNUCU YÜKÜNÜN TAHMİN EDİLMESİ

ÖZET

Sunucu yükü tahmini kavramı, dağıtık sistemlerde yük dengelemesinde ve yük paylaşımında görülür. Dağıtık sistem uygulamalarında yük tahmini için makine öğrenme yöntemlerinin uygulanması kullanılabilirliği ve performansı artırabilir. Sunucu yükü tahmini için bugüne kadar birçok makine öğrenme yöntemi uygulanmıştır. Bu çalışma, verimli yük dengesi sağlayarak ve ana bilgisayar yük anomalilerini tespit ederken iş yükünü doğru tahmin ederek oyun sunucularının performansını artırmaya odaklanmaktadır. Tahmin için Naif Bayes, Genelleştirilmiş Doğrusal Model, Lojistik Regresyon, Hızlı Büyük Marj, Konvolüsyonel Sinir Ağı, Karar Ağaçları, Random Forest, Gradyan Arttırılmış Ağaçlar ve Destek Vektör Makinesi içeren bir model kurulmuştur. Eğitim aşamasında kullanılan veriler, veri aktarımı ve ağ kullanımı miktarının kapsamlı bir analizi yapılarak üretilmiştir. Analiz aşamasında, kesin kaynak gereksinimlerini ortaya çıkarmak için iyi verimlilik göz önünde bulundurulmuştur. Yüksek doğrulukta performans analizi için çeşitli koşullar altında kapsamlı simülasyonlar gerçekleştirilmiştir. Deneyler, sonuçlarda ortaya çıkan algoritmanın, literatürde bulunan diğer algoritmalara kıyasla yük tahmini açısından ümit verici bir performans sunduğunu göstermektedir.

Anahtar Kelimeler: *Makine Öğrenmesi, Sunucu Yükü Tahmini, Oyun Sunucuları, Sınıflandırma, Özellik Çıkarımı, Veri Ön İşleme*

PREDICTING GAME SERVER LOAD BY USING MACHINE LEARNING ALGORITHMS

ABSTRACT

The concept of server load estimation is seen in load balancing and load sharing in distributed systems. Applying machine learning methods for load estimation in distributed system applications can improve availability and performance. To date, many machine learning methods have been applied for server load estimation. This study focuses on improving the performance of game servers by providing efficient load balance and accurately predicting workload while detecting host load anomalies. For the estimation, a model including Naive Bayes, Generalized Linear Model, Logistic Regression, Fast Large Margin, Convolutional Neural Network, Decision Trees, Random Forest, Gradient Enhanced Trees and Support Vector Machine were established. The data used in the training phase was produced through a comprehensive analysis of the amount of data transfer and network usage. In the analysis phase, goodput was taken into consideration to reveal the exact resource requirements. Comprehensive simulations were performed under various conditions for high accuracy performance analysis. The results of the experiments show that the algorithm obtained in the results offers a promising performance in terms of load estimation compared to other algorithms found in the literature.

Keywords: *Machine Learning, Server Load Prediction, Gaming Servers, Classification, Feature Extraction, Data Preprocessing*

I. GİRİŞ

Makine Öğrenmesi (Machine Learning - ML), heterojen veri setleri ile birlikte çalışabilmesi nedeniyle gün geçtikçe önem kazanmaktadır. ML algoritmaları doğrudan veriden öğrenir, gizli bilgileri üretir ve gelecekteki sonuçları öğrenme temelinde tahmin edebilir (Batra & Sachdeva, 2016). Tahminler sınıflandırma veya regresyon yaklaşımı ile yapılabilir. Tahminlerin sınıflandırma doğruluğu, verilerin kalitesine bağlıdır. Çeşitli kaynaklardan üretilen veriler eksik, gürültülü, tutarsız, hacimli ve sınıf dengesiz olabilir (David ve Shwartz, 2014). Bu kusurlu verileri daha fazla analiz için temizlemek ve hazırlamak, veri hazırlama aşamasını gerektirir (Dorian, 2006). Veri kalitesinin elde edilmesi için, makine öğrenmesi, veri ön işleme adı verilen en anlamlı adımlardan birini gerektirir. Bu adım genellikle önemli miktarda zaman alır ve genel model performansını iyileştirmek için dikkatli bir şekilde uygulanmalıdır (Duggal S. ,vd. 2016). Bu çalışmada, oyun sunucu yüklerinin tahmin edilmesi için makine öğrenmesi yöntemlerinin kullanılması ve bu algoritmalarının verdiği sonuçlarının karşılaştırılması amaçlanmıştır. Bu karşılaşmanın sağlıklı yürütülebilmesi ve tahmin seviyesinin yüksek olması için veri ön işleme metotlarından faydalanılmıştır. Eksik veriler tamamlanmış, tutarsız veriler düzeltilmiş, özellik çıkarımı yapılabilmesi için uygun hale getirilmiştir. Makine öğrenmesi yapılacak olan her veride uygulanması beklenen bu metotlar sonucun iyileştirilmesini, eğitim süresinin düşürülmesini ve maliyet açısından optimum değerlere ulaşılmasını sağlayacaktır. Bunların yanında verilere düşük varyans filtresi, yüksek korelâsyon filtresi ve temel bileşenler analizi uygulanmıştır. Oyun sunucu yüklerinin doğru bir şekilde tahmin edilebilmesi, hem oyunun kalitesi, hem bu oyunun kullanılabilirliğinin kalitesini arttıracak, aynı zamanda bu sunucu yüklerinin elektrik kullanımının en uygun hale getirilip operasyonel hataların düşürülmesini sağlayarak gereğinden fazla elektrik harcamayıp maliyet açısından uygun hale getirecektir. Nüfusun artmasıyla birlikte elektrik talebi hızla arttığından, bu talebi verimli bir şekilde yönetmek için akıllı şebekeler kullanılmaktadır. Bu akıllı şebekelerde talep tarafı yönetiminin ana özelliği, akıllı şebekenin operatörlerinin

verimli ve etkili kararlar vermelerine olanak sağlaması nedeniyle yük tahminidir. Üç farklı yük tahmini kategorisi vardır, bunlardan ilki kısa vadeli yük tahmini (Short Term Load Forecasting - STLF, birkaç saat ila birkaç gün arasında değişmektedir), ardından orta vadeli yük tahmini (Mid Term Load Forecasting - MTLF, birkaç gün ila birkaç ay arasında) ve son olarak uzun vadeli yük tahminidir (Long Term Load Forecasting - LTLF, bir yıla eşit veya daha büyük). Bu çalışma, tahmin mekanizmalarının her durumda iyi çalışabilmesi için sahip olunan veriden ve makine öğrenme algoritmalarının gücünden faydalanacaktır. Bahsi geçen üç farklı tahmin için de veriler toplanmış ve birleştirilmiştir. Bu sayede operasyon sırasında, yakın zamanlı ve geleceğe yönelik çalışmalarda planlı ve programlı hareket etmeyi sağlayacaktır. Yük tahmini, çok değişkenli ve çok boyutlu bir tahmin problemidir ki burada sayısal yöntemler kullanarak eğri uydurma (curve fitting) gibi tahmin yöntemleri, rastgele trendleri doğru bir şekilde izlemekte başarısız oldukları için doğru sonuçlar vermez ki bu da makine öğrenmesi algoritmalarının gerekliliğini göstermiştir.

A. Problem

Bu tezde, cevaplanmak istenilen sorunlar şunlardır:

- Oyun Sunucularının yükü tahmin edilebilir mi?
- Bu tahmin, makine öğrenmesi yöntemleri kullanılarak gerçekleştirilebilir mi?
- Kullanılacak veri seti üzerinden, hangi makine öğrenmesi daha iyi bir tahmin yürütebilir?

1. Motivasyon

Devamlı gelişen teknolojiler ve donanımlar, bu donanımları kullanarak geliştirilen oyunlar ve bu oyunların oynandığı çeşitli platformlar tasarlanırken ve yapılandırılırken karşılaşılan en önemli sorunlardan biri olan sunucu yükü problemidir. Oldukça fazla insanın eğlenmek veya başka amaçlarla katıldığı bu oyunlar, iyi kontrol edilmez ve yük eşit dağıtılmaz ise bir kâbusa dönüşebilir ki bu hem memnuniyetsizlik hem de mâli kayıp yaratır. Bundan yola çıkarak yüklerin durumuna göre değişebilen sistemler getirilebilir (Örneğin shard mekanizması, layer mekanizması) ama bunun da etkin bir şekilde yapılması için yükün doğru tahmin

edilebilmesi gereklidir. İşte bu tahminin nasıl yapılacağı ve en iyi sonucu hangisinin vereceği sorunu, bu yüksek lisans tezinin motivasyonunu oluşturmaktadır.

2. Hedefler

Bu tez çalışmasında ulaşılması planlanan hedefler aşağıdaki gibidir:

- Veri setinin tahsis edilmesi ve veri ön işlemesi
- Seçilen makine öğrenme metotlarının açıklanması ve veri setinde uygulanması
- Veri setinin her bir algoritma için ayrı eğitim ve sonucunun çıkartılması
- Tahmin sonuçlarının karşılaştırılarak sahip olunan veri setinde hangi algoritmanın daha iyi performans sunduğunun gösterilmesi
- Bu tahmini arttırmak ve gerçek hayata geçirmek için yapılabilecekler

3. Taslak

Bölüm I'de tez'e giriş, problem motivasyon ve hedefler anlatılmaktadır. Bölüm II'de literatür taraması, yapay zeka ve makine öğrenmesi, veri filtrelemesi ve öznitelik seçimi, ardından sınıflandırma ve tahmin, istatiksel sınıflandırma yöntemleri ve yapay sinir ağları ve son olarak sunucu yükü tahmini probleminden bahsedilecektir. Bölüm III'de materyal ve metotlar, verinin hazırlanması ve filtrelenmesi ve kullanılan metodoloji, ardından makine öğrenmesi yöntemleri anlatılacaktır. Bölüm IV'de bulgular ve tartışma, V. bölümde ise sonuçlar ve gelecek araştırmalara yönelik öneriler verilecektir.

II. LİTERATÜR TARAMASI

Yapay zekâ ve makine öğrenmesi tekniklerinin geçmişten günümüze nasıl taşındığı ve ne gibi gelişmeler olduğu, hangi büyük firmalar tarafından ne amaçlarla kullanıldığı anlatılmıştır. Ayrıca tezin içerisinde kullanılan algoritmaların tarihinden de bahsedilerek güncel örnekler verilmiş, bu konular hakkında yapılan ayrıntılı literatür taramasından bahsedilmiştir. Konular, kronolojik sıraya göre verilmiştir.

A. Yapay Zekâ ve Makine Öğrenmesi

Yapay zekâ olarak adlandırılan teknolojinin ilk adımlarını, 1936 yılında Alan Turing atmıştır. İngiliz matematikçi Alan Turing, teorilerini, "Turing makinesi" olarak bilinen bir bilgisayar makinesinin, birden fazla bireysel adımda parçalanıp bir algoritma ile temsil edilmeleri koşuluyla bilişsel süreçleri uygulayabileceğini kanıtlamak için kullanmıştır (Turing, 1936). 1956 yazında, bilim adamları New Hampshire'daki Dartmouth College'da bir konferans için toplandıklarında, insan zekâsının diğer özelliklerinin yanı sıra öğrenmenin özelliklerinin de makinelerle simüle edilebileceğine inanıyorlardı. Programcı John McCarthy, bu durumun yapay zekâ olarak adlandırılmasını önermiştir. Konferansta, bir düzine matematik teoremini ve verisini kanıtlamayı başaran, dünyanın ilk AI programı olan mantık teorisidir (Url-18). 1966 yılında Alman Teknoloji Enstitüsü'nden Alman Amerikalı bilgisayar bilimcisi Joseph Weizenbaum, insanlarla iletişim kuran bir bilgisayar programı geliştirmiştir. ELIZA, psikoterapist gibi çeşitli konuşma ortaklarını taklit etmek için komut dosyalarını kullanmıştır. Weizenbaum, ELIZA'nın bir insan konuşma ortağı yanılsaması yaratması için gerekli araçların basitliğinden etkilendiğinden bahsetmiştir (Url-16). 1972 yılında MYCIN ile yapay zekâ tıbbi uygulamalara giriş yapmıştır. Stanford Üniversitesi'nde Ted Shortliffe tarafından geliştirilen uzman sistem, hastalıkların tedavisinde kullanılmıştır. Uzman sistemler, uzman alan için bilgiyi formüller, kurallar ve bilgi veri tabanı kullanarak birleştiren bilgisayar programlarıdır ve tıpta tanı ve tedavi desteği için kullanılırlar (Url-3). 1986 yılında bilgisayara ilk defa bir ses verilmiştir. Terrence J. Sejnowski ve Charles Rosenberg,

'NETtalk' programlarını, örnek cümleler ve fonem zincirleri girerek konuşmayı öğretmişlerdir. NETtalk kelimeleri okuyabilme ve doğru telaffuz edebilme, bilmediği kelimelere de öğrendiklerini uygulayabilme yeteneğine sahiptir. Bu, büyük veri setleriyle birlikte verilen ve bu temelde kendi sonuçlarını çıkartabilen programlar olan yapay sinir ağlarından biridir. Yapay sinir ağlarının bu yapıları ve işlevleri insan beynine benzerlik göstermektedir (Sejnowski ve Rosenberg, 1986). 1997 yılında IBM'den AI satranç bilgisayarı 'Deep Blue' görevdeki satranç dünyası şampiyonu Garry Kasparov'u bir turnuvada yenmiştir. Bu, daha önce insanların egemen olduğu bir alanda tarihi bir başarı olarak kabul edilmiştir. Ancak eleştirmenlerin Deep Blue hakkında buldukları hata, Deep Blue'nun bilişsel zekâdan ziyade tüm olası hareketleri hesaplayarak hareket etmesine yapay zekâ demenin ne kadar doğru olacağıdır (Url-9). Donanım ve yazılım alanlarındaki teknolojik sıçramalar, yapay zekânın günlük hayata girmesinin önünü açmıştır. Bilgisayarlardaki, akıllı telefonlardaki ve tabletlerdeki güçlü işlemciler ve grafik kartları, tüketicilerin AI programlarına düzenli erişimini sağlamıştır. Özellikle dijital asistanlar büyük popülerliğe ulaşmışlardır: Apple'ın 'Siri'sı 2011'de pazara giriş yapmış, Microsoft 2014'te Cortana'yı piyasaya sürmüş ve Amazon, Amazon Echo'yu 2015'te 'Alexa' adlı ses servisi ile seslendirmiştir (Url-4; Url-24; Url-22). 2011 yılında "Watson" adlı bilgisayar programı, bir ABD televizyon yarışması programında animasyonlu bir ekran sembolü şeklinde yarışmıştır ve insan oyunculara karşı kazanmıştır. Bunu yaparken, Watson doğal dili anladığını ve zor sorulara hızlıca cevap verebildiğini kanıtlamıştır (Url-23). Bu iki örnek yapay zekânın yeteneklerini ortaya koymuştur. 2018 Haziran ayında, IBM'den 'Project Debater', iki ana tartışmacıyla karmaşık konuları tartışmış ve oldukça iyi bir performans göstermiştir. Google bir konferansta AI programında 'Duplex'in bir kuaföre nasıl telefon açtığını ve nasıl randevu aldığını, bunu hattın diğer ucundaki hanımefendi'ye makine olduğunu fark ettirmeden yaptığını göstermiştir (Url-7). Tüm bunlara rağmen yapay zekâ, on yıllarca süren araştırmalara rağmen henüz başlangıç aşamasındadır. Otonom sürüş veya ilaç gibi hassas alanlarda kullanılmadan önce dışarıdan müdahalelere karşı daha güvenilir ve güvenli olması gerekir. Bir başka amaç da AI sistemlerinin kararlarını açıklamayı öğrenmesidir, böylece insanlar onları kavrayabilir ve AI'ın nasıl düşündüğünü daha iyi araştırabilir.

Makine öğrenmesi adına ilk adım, 1943'te nörofizyolog Warren McCulloch ve matematikçi Walter Pitts'in nöronlar ve nasıl çalıştıkları hakkında bir yazı yazmasıdır. Bir elektrik devresi kullanarak bunun bir modelini yaratmaya karar vermişlerdir ve bu sayede yapay sinir ağları ortaya çıkmıştır (McCulloch ve Pitts, 1943). 1950 yılında, Alan Turing dünyaca ünlü Turing Testini geliştirmiştir. Bu testi bir bilgisayarın geçmesi için, bir insanı, bir bilgisayar değil de bir insan olduğuna ikna edebilmesi gerekir (Turing, 1950). Çalıştığı gibi öğrenebilecek ilk bilgisayar programı, 1952 yılında Arthur Samuel tarafından geliştirilmiş dama oynayan bir oyundur ve bu 1959 yılında IBM Journal tarafından yayınlanmıştır (Samuel, 1959). Frank Rosenblatt, 1958'de Perceptron adlı ilk yapay sinir ağını tasarlamıştır: Bu perceptron'un temel amacı model ve şekil tanıma olmuştur (Rosenblatt, 1958). Sinir ağının son derece erken bir örneği de 1959'da Bernard Widrow ve Marcian Hoff'un Stanford Üniversitesi'nde iki model yarattıkları zaman gelmiştir. İlki ADELINE olarak adlandırılıyordu ve ikili kalıpları tespit edebiliyordu. Örneğin, bir bit akışında, bir sonrakinin ne olacağını tahmin edebilmekteydi. Yeni nesil MADELINE olarak adlandırıldı ve telefon hatlarında yankıyı ortadan kaldırabiliyordu ki bu teknoloji günümüzde halen kullanılmaktadır. MADELINE'ın başarısına rağmen, 1970'lerin sonlarına kadar birçok nedenden dolayı, özellikle de Von Neumann mimarisinin popülaritesi gibi pek fazla ilerleme olmamıştır (Url-6). Bu mimari, talimatların ve verilerin aynı hafızada saklandığı, sinir ağlarından daha anlaşılması kolay olan bir mimaridir ve pek çok insan buna dayanarak programlar geliştirmiştir. 1982'de John Hopfield, nöronların gerçekte nasıl çalıştığına benzer bir şekilde çift yönlü hatları olan bir ağ kurmayı önerdiğinde, sinirsel ağlara olan ilgi yeniden artmaya başlamıştır (Hopfield, 1982). Ayrıca 1982'de Japonya, Amerikan finansmanını bölgeye teşvik eden ve böylece bölgede daha fazla araştırma yapan daha gelişmiş sinir ağlarına odaklandığını açıklamıştır (Url-12). Sinir ağları geri yayılımı kullanır ve bu önemli adım 1986'da Stanford psikoloji bölümünden üç araştırmacı, Widrow ve Hoff tarafından 1962'de oluşturulan bir algoritmayı genişletmeye karar verdiğinde ortaya çıkmıştır (Rumelhart vd. , 1986). Bu, sinir ağında birden fazla katmanın kullanılmasına izin vermiştir ki öğrenme süresi uzun sürecek olan 'yavaş öğrenenler' olarak bilinir. 1980'lerin ve 1990'ların sonları makine öğrenme adına çok büyük gelişmelere tanık olmamıştır. Ancak 1997 yılında, satranç oynayan bir bilgisayar olan IBM bilgisayarı Deep Blue dünya satranç şampiyonunu yendiğinde, bu gelişmelerin ciddiyeti farkındalık yaratmaya başlamıştır (Url-9). O zamandan beri,

AT&T Bell Laboratories'in rakam tanıma konusundaki araştırmasının 1998'de olduğu gibi ABD Posta Servisi'nden el yazısı ile yazılmış posta kodlarının tespitinde iyi bir doğrulukla sonuçlandığı ki bunda geri yayılım algoritması kullanılıyordu, kaydedilmiştir (Rey, 1983). 21. Yüzyılın başından beri birçok işletme, makine öğrenmesinin hesaplama potansiyelini arttıracağını fark etmiştir. Bu nedenle rekabetin önünde kalmak için makine öğrenmesi adına oldukça yoğun araştırma yapmaya devam etmektedirler. Bu projelerden bazıları:

GoogleBrain: 2012-Google'dan Jeff Dean tarafından oluşturulan ve görüntü ve videolarda kalıp algılamaya odaklanan derin bir sinir ağıdır. Google'ın kaynaklarını kullanabıldiği için bu da onu daha küçük sinir ağlarıyla karşılaştırılamaz hale getirmiştir. Daha sonra YouTube videolarındaki nesneleri tespit etmek için kullanılmıştır (Url-8).

AlexNet: AlexNet, 2012'de ImageNet yarışmasını büyük bir farkla kazanmıştır; bu da GPU'ların ve Konvolüsyonel Sinir Ağlarının makine öğreniminde kullanılmasını sağlamıştır. Ayrıca CNN'lerin verimliliğini büyük ölçüde artıran bir aktivasyon işlevi olan ReLU'yu yaratmışlardır (Krizhevsky, vd. 2012).

DeepFace: Facebook'un geliştirdiği ve insanları bir insanın bildiği hassasiyetle tanıyabildiklerini iddia ettikleri bir Derin Sinir Ağıdır (Tagman, vd. 2014).

DeepMind: Bu şirket Google tarafından satın alınmıştır ve temel video oyunlarını insanlarla aynı seviyelerde oynayabileceğini söylemişlerdir. Deepmind 2016 yılında, dünyanın en zor tahta oyunlarından biri olarak kabul edilen Go oyununda profesyonel oyuncuyu yenmeyi başarmıştır (Url-19).

OpenAI: Elon Musk ve diğer ekip arkadaşları tarafından, insanlığa fayda sağlayabilecek güvenli bir yapay zekâ oluşturmak için 2015 yılında yaratılan, kar amacı gütmeyen bir organizasyondur (Url-13).

Amazon Machine Learning Platform: Bu, Amazon İnternet Servislerinin 2015 yılında üretilen bir parçasıdır ve çoğu büyük şirketin makine öğrenmesine nasıl katılmak istediğini gösterir. Arama önerileri ve Alexa gibi düzenli olarak kullanılan hizmetlerden Prime Air ve Amazon Go gibi daha deneysel olanlara kadar iç sistemlerinin birçoğunu yönlendirdiğinden bahsetmişlerdir (Url-1).

ResNet: Resnet, evrişimli sinir ağlarında önemli bir gelişme olarak kabul edilmektedir ve 2015 yılında ortaya atılmıştır(Url-2).

U-net: Biyomedikal görüntü segmentasyonunda uzmanlaşmış 2015 de ortaya atılan bir evrişimli sinir ağı mimarisidir (Url-20). Amazon Web Hizmeti Makine Öğrenme platformlarını güçlendirmek için Nvidia kullanılmıştır. Bunun nedeni, özellikle makine öğrenmesi için GPU'lar oluşturmalarıdır, örnek olarak Mayıs 2017'de ilan edilen Tesla V100 gösterilebilir (Url-11). Makine öğrenmede matris aritmetiği için kullanılan Tensor Çekirdeklerini kullanmışlardır.

1. Veri Filtreleme

Bu bölümde, veri hazırlama ve filtreleme kısmında kullanılacak olan tekniklerinden bahsedilecektir. Veri filtreleme, gereksiz bilgilerin kaldırılması için satırların yani vakaların elimine edilmesi anlamına gelir. Bu, modellenecek değişkenlerin sinyalini netleştirmek için yapılır. Gereksiz bilgilerin kaldırılması, analiz seviyesinin altındaki gürültüyü azaltır. Sinyal bu şekilde ifade edildiğinde, sinyal işleme yapılıyormuş gibi düşünülebilir; ancak, buradaki sinyal bir radar sinyali değil, veri sinyalidir. Her iki tür sinyal de sadece altta yatan bir bilgi alanının ifadesidir. Bir radar sinyali, altta yatan mesafe ve konum alanının ifadesidir. Bir uydu görüntüsü sinyali görsel bir alanın bir ifadesidir. Benzer şekilde, kurumsal bir veri tabanındaki müşteri yıpratma sinyali, bir şirketteki müşteri tutma alanının bir ifadesidir.

a. Aykırı Değerlerin Kaldırılması

Aykırı değerlerin üstesinden gelmenin en basit yolu, onları içeren satırları kaldırmaktır. Bazen, aykırı yani anormal değerlerin saklanmasını istemek gerekebilir. Aslında, bazı aykırı değerler, kredi riski, sahtekârlık ve ağ müdahaleleri gibi diğer nadir olayların modellenmesinde birincil ilgi alanına girmektedir. Normal yanıt modellerinde, aşırı aykırı sıranın silinmesi ya da değerin sabit ya da orta ya da ortancaya göre tahmin edilmesiyle çıkarmak iyi bir fikir olabilir. Bunun nedeni, normal yanıtı tanımlamaya yardımcı olan verileri modellemek istenmesidir. Aykırı değerler veri kümesinde bırakılırsa, yalnızca modelin tahmin edilebilirliğini azaltacak şekilde gürültü enjekte ederler. Ancak, tüm değerler verilerde tutulması gerektiğine itiraz edilebilir, çünkü modelin üretim operasyonlarda bunun gibi değerlerin puanlanması gerekir. Hawkins, aykırılığı farklı bir mekanizma tarafından yaratıldığı şüphesini uyandıran diğer gözlemlerden çok sapan bir gözlem olarak tanımlamıştır. Hawkins, dört çeşit aykırı algılama algoritmasını tartışmaktadır (Hawkins, 1980):

• Kritik mesafe ölçütlerine dayalı olanlar

• Yoğunluk ölçütlerine dayalı olanlar

• Projeksiyon özelliklerine dayalı olanlar

• Veri dağıtım özelliklerine dayalı olanlar

Bazı veri madenciliği paketlerinde aykırı değerlerin belirlenmesi için özel yordamlar vardır. Örneğin, Statistica Data Miner, belirli bir değer aralığının veya frekans dağılımının oranının ötesinde aykırı değerlerin otomatik olarak kontrol edilmesini ve kaldırılmasını sağlayan bir tarif modülü sağlar. Bu araçtaki dağıtım aykırı seçeneğinin kullanılması, kuyruklardaki durumları ortalama değerden uzaklık için güven aralığının ötesinde kesecektir. Zaman serisi verilerini analiz ederken daha karmaşık filtreleme gerekli olabilir. Ve zaman serisi verilerinin analizinde iletim ve görüntü sinyallerinin sinyal işlemesi için en yakın analojileri görülür (Url-15).

b. Zaman Serisi Filtreleme

Zaman serisi verilerinin filtrelenmesinin en iyi uygulamalarından biri Timothy Masters tarafından sağlanmaktadır (Masters, 1995). Masters, filtrelemenin ne olduğu ve zaman serisi verilerinin modellenmesine yardımcı olmak için nasıl etkili bir şekilde kullanılabileceği konusunda sezgisel açıklamalar sağlamıştır. Sinyal filtreleri, aralığın tepesindeki, aralığın altındaki veya her ikisindeki yüksek frekanslı sinyal dalgalanmalarını yani titreşimleri kaldırır. Önce alçak geçişli bir filtre uygulanır ki bu verileri belirtilen en yüksek kabul edilebilirlik seviyesinin altına iletir. Ardından yüksek geçiş filtresi uygulanır bu filtre ise alçak geçiş filtresinin tersi işlemini yapar; verileri belirlenmiş bir düşük kabul edilebilirlik seviyesinin üstüne çıkarır. Son olarak ise bant geçiş filtresi çalışır bu da yalnızca düşük bir değerin üzerindeki ve yüksek bir değerin altındaki verileri iletir.

Çoğu veri kümesi zaman serisi değil modelleme için hazırlanmalıdır. Bunun nedeni, sinyal işleme tekniklerinin zaman serisi verileri dışındaki veri kümelerine çok etkili bir şekilde uygulanabilmesidir. Bu teknikler, bir faturalandırma sistemlerinden gelen zaman serisi verilerini analiz etmek ve kayıtları satın almak için kullanılabilir. Bir tahmin değişkeni, bir hedef değişkeninin öngörüsündeki katkısı bağlamında görüldüğünde, bu katkı oranını hedef değişkenin durumunun veya seviyesinin bir

sinyali olarak düşünebiliriz. Bazı değişkenler daha güçlü bir sinyal sağlayabilir ki bu daha tahmin edilebilir olmasını sağlar ve diğer değişkenler daha az olabilir. Öngörü değişkeninin değerlerinde belirli bir miktar gürültü yani hedef sinyalde karışıklık olacaktır. Zaman gürültüsü sinyal filtrelemesine çok benzer şekillerde bu gürültünün bir kısmı seçici olarak kaldırılabilir. Alçak geçişli bir veri filtresi, belirli bir eşik değerin altındaki değerlere sahip durumları ortadan kaldırarak uygulanabilir. Bu işlemin etkisi modelleme algoritmasına önemsiz girdileri kaldırmak olacaktır. Öte yandan, yüksek eşikli bir filtre, bir eşiğin üstündeki durumları yani aykırı değerleri kaldırmak için kullanılabilir. Veri madenciliğinde var olan önemli noktalardan biri, bu operasyonlar için doğru eşikleri seçmektir ki bunu doğru yapmak için veri alanını çok iyi bilmek gerekir.

c. Öznitelik Seçimi

Öznitelik seçimi üzerine araştırmalar son yıllarda oldukça aktif bir şekilde devam etmiştir (Kira ve Rendell, 1992; Koller ve Sahami, 1996; Kohavi ve John, 1997; Pupil ve Novovicova, 1998; Chapellle, vd. 2002; Gilad-bachrach, vd. 2004; Hilario ve Kalousis, 2008). Bu bölüm var olan algoritmaları kısaca gözden geçirir ve önceki çalışmalarla ilgili bazı önemli konuları tartışır. Mevcut algoritmalar, konuyla ilgili öznitelikleri aramak için kullanılan kıstaslar açısından geleneksel olarak sarıcı yani wrapper veya filtre yöntemleri olarak kategorize edilebilir (Kohavi ve John, 1997). Sarıcı yöntemlerde, seçilen bir öznitelik alt kümesinin doğruluğunu değerlendirmek için bir sınıflandırma algoritması kullanılırken, filtre yöntemlerinde kıstas işlevleri öznitelik alt kümelerini oluşum içi içerikleriyle, tipik olarak sınıflar arası mesafeleriyle örneğin fisher skoru veya istatistiksel ölçümler kullanır (ör. Herhangi bir özel öğrenme algoritmasının performansını doğrudan optimize etmek yerine, t-testinin p değeri). Bu nedenle, filtre yöntemleri hesaplama olarak çok daha verimlidir, ancak genellikle sarma yöntemleri kadar iyi performans göstermezler.

Sarıcı yöntemlerle ilgili en büyük sorunlardan biri, çok sayıda sınıflandırıcı yetiştirme gereksinimi nedeniyle yüksek işlem karmaşıklıklarıdır. Bu sorunu gidermek için birçok sezgisel algoritma (örneğin ileri ve geri seçimi (Pudil ve Novovicova, 1998)) önerilmiştir. Bununla birlikte, sezgisel niteliklerinden dolayı, hiçbiri uygun değer garantisi veremez. Gen ekspresyonunda mikro dizi veri analizinde olduğu gibi, on binlerce özellik ile bir hibrit yaklaşım genellikle kabul edilir, burada özellik sayısı önce bir filtre metodu kullanılarak azaltılır ve daha sonra

azaltılmış özellik setine bir sarma metodu uygulanır. Bununla birlikte, sarma yönteminde kullanılan sınıflandırıcıya bağlı olarak, aramayı gerçekleştirmek halâ birkaç saat sürebilir. Karmaşıklığı azaltmak için pratikte basit bir sınıflandırıcı (örneğin doğrusal sınıflandırıcı), özellik altkümelerinin iyiliğini değerlendirmek için sıklıkla kullanılır ve seçilen özellikler daha sonra veri analizinde daha karmaşık bir sınıflandırıcıya beslenir. Bu özellik, öznitelik çıkarımının ortaya çıkmasına neden olur - bazı durumlarda bir sınıflandırıcı için en uygun olan bir öznitelik altkümesi diğerleri için iyi çalışmayabilir (Hilario ve Kalousis, 2008). Bir sarıcı yöntemiyle ilişkilendirilen bir başka sorun, çok sınıflı sorunlar için özellik seçimi gerçekleştirme kabiliyetidir. Büyük ölçüde, bu özellik, çok sınıflı problemlerin üstesinden gelmek için bir sarıcı yönteminde kullanılan bir sınıflandırıcının yeteneğine bağlıdır. Pek çok durumda, bir adet çoklu sınıf problemi ilk önce bir hata-doğru kod yöntemi (Dietterich ve Bakiri, 1995; Sun, vd. 2005) kullanılarak birkaç ikili probleme ayrıştırılır ve daha sonra her ikili problem için öznitelik seçimi yapılır. Bu strateji, bir sarıcı yönteminin hesaplama yükünü daha da arttırır. Literatürde nadiren değinilen bir konu algoritmik uygulamadır. Birçok sarıcı metodu, çok sayıda sınıflandırıcının eğitilmesini ve birçok parametrenin manuel olarak tanımlanmasını gerektirir. Bu onların uygulanmasını sağlar ve makine öğreniminde oldukça karmaşık, talepkâr bir uzmanlık kullanır. Bu, muhtemelen filtre yöntemlerinin biyomedikal toplumunda daha popüler olmasının ana nedenlerinden biridir (Veer, 2002; Wang, 2005).

Yukarıda belirtilen sorunları doğrudan sarıcı yöntemi çerçevesinde ele almak zordur. Bu zorluğun üstesinden gelmek için, gömülü yöntemler son zamanlarda artan bir ilgi görmüştür (Weston, vd. 2001; Ng, 2004; Guyon ve Elisseeff, 2003; Chapelle, vd. 2006; Lal, vd. 2006; Guyon, vd. 2002; Zhu, vd. 2004). Gömülü yöntemler, bir sınıflandırıcının öğrenme sürecine öznitelik seçimini içerir. Bir öğrenme sürecindeki özelliklerin önemini belirtmek için genellikle ikili değerler yerine gerçek değerli sayıları kullanan bir özellik ağırlıklandırma stratejisi benimsenir ki bu stratejinin birçok avantajı vardır. Örneğin, ilgili özelliklerin sayısının belirlenmesine gerek yoktur. Ayrıca, bir kombinasyonel araştırmayı önlemek için standart optimizasyon teknikleri (örneğin, dereceli alçalma) kullanılabilir. Bu nedenle, gömülü yöntemler genellikle sarma yöntemlerinden daha hesaplamalıdır. Yine de, hesaplamaların karmaşıklığı, özniteliklerin aşırı derecede fazla olması durumunda önemli bir konudur. Algoritma uygulaması, özniteliklerin dışa aktarılabilirliği ve çok sınıflı

sorunlara genişletme gibi diğer konular da devam etmektedir. Yakın zamanda geliştirilen bazı gömülü algoritmalar, belirli varsayımlar altında büyük ölçekli özellik seçimi sorunları için kullanılabilir. Örneğin, (Weston, vd. 2001; Chapelle, vd. 2002) doğrudan SVM formülasyonunda şekillendirme özelliği seçimi önerir; burada ölçeklendirme faktörleri, hata oranına teorik bir üst sınır gradyanı kullanılarak ayarlanır. RFE (Guyon, vd. 2002), özellikle mikro-dizi veri analizi için tasarlanmış iyi bilinen bir özellik seçim yöntemidir. Geçerli bir dizi özelliğe sahip bir SVM sınıflandırıcısını yinelemeli olarak eğiterek ve ardından küçük özellik ağırlıkları olan özellikleri buluşsal olarak kaldırarak çalışır. Sarma yöntemlerinde olduğu gibi, SVM'nin yapısal parametrelerinin, örneğin yinelemeler sırasında çapraz doğrulama kullanılarak yeniden tahmin edilmesi gerekebilir. Ayrıca, doğrusal bir çekirdek genellikle hesaplama nedenleriyle kullanılır. l1-SVM ile doğrusal bir çekirdeğe (Zhu, vd. 2004), uygun bir parametre ayarlamasıyla, sadece ilgili özelliklerin sıfır ağırlık almadığı yerlerde, seyrek bir çözüme yol açabilir. Benzer bir algoritma l1 düzenlemesi olan lojistik regresyondur. (Ng, 2004) tarafından l1 düzenli lojistik regresyonun özelliklerin sayısına göre logaritmik bir örnek karmaşıklığına sahip olduğu kanıtlanmıştır. Ancak, bu yaklaşımlardaki veri modellerinin doğrusallık varsayımları genel sorunlara uygulanabilirliklerini sınırlamaktadır.

B. Sınıflandırma ve Tahmin Algoritmaları

Makine öğrenimi ve istatistikte sınıflandırma, bilgisayar programının kendisine verilen veri girişinden öğrendiği ve daha sonra yeni gözlemleri sınıflandırmak için bu öğrenmeyi kullandığı denetimli bir öğrenme yaklaşımıdır. Bu veri seti basitçe iki sınıf olabilir (kişinin erkek mi kadın mı olduğu, postanın spam mı yoksa spam olmayan mı olduğu gibi) veya çok sınıflı da olabilir. Bazı sınıflandırma problemlerine konuşma tanıma, el yazısı tanıma, biyometrik tanımlama, doküman sınıflandırma örnek gösterilebilir. Bu bölümde istatistiksel sınıflandırma algoritmalarından, yapay sinir ağlarından ve sunucu yükü tahmini probleminden bahsedilecektir.

1. İstatistiksel Sınıflandırma Algoritmaları

İstatistiksel sınıflandırmanın çok çeşitli uygulamaları bulunmaktadır. Örneğin X-ışını mamografisini kullanarak bir tümörün iyi huylu veya habis olarak teşhisi, bir sınıflandırma görevidir ki burada sınıfları iyi ve kötü huylu tümörler oluşturur (Remes ve Haindly, 2015). Ayrıca optik karakter tanıma (OCR) problemi de buna bir örnektir (Sultane, vd. 2017). OCR'daki görev el yazısı harflerinin resimlerini tanımak ve bunları yirmi dört sınıftan birine atamaktır. Bir e-postanın spam veya tehlikeli olarak öngörülmesi (Jiddiga ve Sammulal, 2013), bilgisayarlı görüde yüz tespiti (Zhang, vd. 2017), kredi kartı işlemlerinde sahtekârlık tespiti (Naik ve Kanikar, 2019) ve biyoinformatikte mikrodizi verilerinin sınıflandırılması (Hameed, vd. 2018) iyi bilinen sınıflandırma örnekleridir. Sınıflandırma görevi şu şekilde tanımlanabilir: Bir örnek, bir dizi özellik ve bir sınıf etiketi (kategori) ile temsil edilen bir nesne olsun. Bir öğrenme algoritması (sınıflandırıcı) esasen özelliklerden bir sınıf etiket setine kadar bir eşleme işlevidir. Denetimli makine öğrenmesi bağlamında, sınıflandırma görevi, etiketlenmiş durumlardan oluşan bir veri kümesinin varlığını üstlenir. Etiketli veriler üzerinde sınıfları otomatik olarak öngöreceği ve gelecekteki durumlar için kullanılabileceği şekilde bir sınıflandırıcı oluşturulur. Sınıflandırma görevi, etiketli veri setinin getirdiği kısıtlamalara tabi olarak, olası tüm haritalama fonksiyonlarından bir sınıflandırıcı aramaktır.

2. Yapay Sinir Ağları

Yapay Sinir Ağları, beyindeki nöronların biyolojik modelinden esinlenen doğrusal olmayan bir makine öğrenme algoritmaları sınıfıdır (Bishop, 1995). İki ana denetimli öğrenme görevini, sınıflandırma ve regresyonu gerçekleştirebilen karmaşık modellerdir. Yapay sinir ağları, yapılandırılmış veri, görüntü, ses vb. farklı veri biçimlerinden öğrenme yeteneğine sahiptir. Yapay sinir ağları özellikle biçim tanıma görevlerini yerine getirmede iyidir; dolayısıyla, uygulamalarının çoğu bu alandadır. Yapay sinir ağlarının güçlü yönlerinden biri, girdi verileri hakkında herhangi bir varsayımda bulunmadıkları, ancak girdiler ve çıktılar arasındaki ilişkiyi örnek olarak öğrenmeye çalıştıklarıdır. Bu, insan beyninin nasıl öğrendiğine benzerlik göstermektedir. Dolayısıyla, yapay sinir ağları, girdiler ve çıktılar arasındaki temel ilişkinin bilinmediği problem ortamlarında çok faydalıdır. Sinir sisteminin çalışma prensibi ve insan davranışları baz alan çalışmalar (McCulloch ve Pitts, 1943; Hebb, 1949; Russel ve Norvig, 1995), perseptron çalışmaları (Minsky ve Papert, 1988;

Smelser ve Baltes, 2001), geri yayılım algoritması (Rumelhart, vd. 1986; Werbos, 1994), boyutsal azaltma (Hinton ve Salakhutdinov, 2006), çekirdek yani kernel makinesi çalışmaları (Bottou, vd. 2007), ses tanıma (Url-26), GPU kullanımına başlanması (Raina, vd. 2009), el yazısı tanıma (Ciresan, vd. 2010), Google ve IBM ekibinin derin sinir ağı çalışmaları (Ciresan, vd. 2010), Google Search (Url-10), ileri beslemeli yapay sinir ağları (Glorot ve Bengio, 2010), ileri beslemeli yapay sinir ağlarının veri büyüklüğü ile ilgili sorunlara yeni bir bakış açısı (Hinton, vd. 2012) ve evrişimli sinir ağları (Krizhevsky, 2012), yapay sinir ağlarının kullanıldığı geniş çerçeveye örnek olarak gösterilebilir.

C. Sunucu Yükü Tahmini Problemi

Son yıllarda, internet çok hızlı büyüdükçe, sunucular ve sistemler gittikçe daha karmaşık hale gelmiştir ve bu nedenle performanslarını kontrol etmek ve dengelemek daha da önemli bir sorun olmaya başlamıştır. Veri işleme sunuculara odaklanmaya başladığında, sunucu yükü katlanarak artmıştır ve performans yönetimindeki diğer zorluklar başlamıştır. Mevcut iş kaynaklarını en uygun düzeyde ve israf etmeden kullanırken, performans iş yükünü (kullanıcı istekleri - iş yükü) verimli ve doğru bir şekilde tahmin etmek çok önemlidir. Doğru tahminler, sunucuların performansını artıracak ki bu da kullanıcı memnuniyetini sağlayacaktır, bunun ardından bu tahminlerle kurulacak bir sistem ve acil durum mekanizması oluşturularak riskler ve kayıplar en aza indirilecektir.

Sunucular, diğer tüm elektronik cihazlar gibi enerji tüketir. Uzun süre yüksek performansla çalıştıklarında, çok fazla elektrik tüketirler ve gereksizce kaynakları boşa harcarlar. Bu yüksek performansı kontrol etmek çok fazla iş yükü yaratır ve basit bir hata bile soğutma sorunları, bant genişliğinde azaltma, yüksek gecikme süresi, ani yükselmeler ve daha fazlası gibi bir dizi problem yaratabilir. Tüm bu sorunlar bir araya geldiğinde, kaçınılmaz görünen ve üstesinden gelinemeyen bir sorun ortaya çıkabilir. Ponemon Institute tarafından yürütülen ve Emerson Network Power tarafından desteklenen bir çalışmaya göre, tek bir veri merkezi kesintisinin ortalama maliyeti 700.000 ila 900.000 dolar arasındadır ki bu da önemli ölçüde artabilir. Örneğin, Delta Airlines veri merkezi Ağustos 2016'da kısa bir süre hizmet dışı kalsa da, bu olay yaklaşık iki bin uçuşun iptal edilmesine neden olmuş ve şirket üç gün boyunca 150 milyon ABD Doları tutarında zarar görmüştür (Url-5). Farklı

endüstrilerdeki veri merkezi kesintilerinin maliyetlerinin analiz edildiği bir çalışmada (Cao, vd. 2018), enerji sektöründe veri merkezi arızası maliyetinin 2,1878 milyon dolar, telekomünikasyon sektöründe 2,0662 milyon dolar ve finans sektöründe 1,4951 milyon dolar olduğu belirtilmektedir. 2012'de, UPTIME Enstitüsü dünya genelinde 94 bilgisayar odasını inceleyerek, toplamda 291 olay ve 8 başarısızlıktan oluşan bir rapor hazırlamıştır (Url-25). 291 olaydan % 39'u aşırı yüklemeden, 8 başarısızlıktan ise aynı şekilde % 25'i operasyonel sorunlardan kaynaklandığını belirtmişlerdir. Bu olaylardan ise sadece % 13' ünün müdahale ile telafi edildiğini ve kurtarıldığını ortaya koymuşlardır.

III. MATERYAL VE METOTLAR

Bu bölümde, seçilen makine öğrenmesi algoritmalarının eğitimi için kullanılacak olan veri setinin nasıl elde edildiğinden, bu veri setinin içeriğinden ve karşılaştırma yapılacak olan algoritmaların metodolojisinden bahsedilecektir.

A. Sunucu Yükü Tahmini Veri Seti

Eğitimde kullanılacak olan veri seti, dünya üzerinde en çok oynanan "MMORPG" oyunlarından (Çok sayıda insanın aynı anda katıldığı çevrimiçi rol yapma video oyunu) birinin veri merkezinden elde edilmiştir. Bu veri merkezinde 9 adet yedek sunucu, 20000 farklı sistem ve dinleyiciler, 13250 ana blade sunucu, 75000 cpu çekirdeği, 38 farklı sınıf ve 16 tane ana sınıftan oluşan bu yapı, gelip giden network trafiğini kontrol edip kayıt altında tutmaktadır. Bu trafikten belirli zaman aralıklarıyla veriler çekilmiş ve birleştirilmiştir. Ortaya çıkan bu veri 6.732.829.683 farklı satırdan oluşmaktadır. Eğitim süresinin kısaltılması ve tahmin sonucunun arttırılması açısından boyutsal küçültme, normalizasyon, düşük varyasyon filtresi ve yüksek korelâsyon filtresi uygulanmıştır. Bu algoritmalar için 33,321 adet model oluşturulmuş ve sonuçlar elde edilmiştir. Verilerin gizliliği ve kötüye kullanımı riskine karşı olarak, bu sınıflar ve asıl verileri paylaşılmamıştır. Normalizasyon uygulandıktan sonra elde edilen sonuçların bir kısmının ekran görüntüsü şekil-1'de gösterilmiştir:

0,215261	0,329169	0,788657	0,950809	0,591771	0,086314	0,804404	0,350844	0,09923	0,040581	0,218576	0,090784	0,93768	0,495147	0,340777	0,304858	0
0,56676	0,080308	0,643996	0,743212	0,405894	0,418093	0,509551	0,098499	0,543358	0,499179	0,218517	0,523121	0,076516	0,55602	0,570004	0,127193	0
0,330631	0,683145	0,952717	0,179944	0,804666	0,950897	0,38575	0,519171	0,561902	0,220528	0,48768	0,612326	0,974306	0,145384	0,987795	0,790208	1
0,756585	0,774024	0,840616	0,541532	0,306879	0,703976	0,241125	0,34574	0,72354	0,636039	0,51089	0,703951	0,673534	0,468951	0,126249	0,386085	1
0,367483	0,082593	0,989835	0,241524	0,283477	0,109487	0,447575	0,844419	0,830352	0,5487	0,466375	0,418409	0,836686	0,690153	0,448588	0,201331	0
0,889774	0,246465	0,540646	0,507769	0,194569	0,767498	0,959164	0,067747	0,962253	0,780889	0,167181	0,272565	0,017232	0,460193	0,503969	0,732176	1
0,291237	0,337557	0,131549	0,89843	0,473981	0,459323	0,512573	0,646956	0,092705	0,353356	0,335358	0,995859	0,077018	0,428451	0,343036	0,073157	0
0,179807	0,991049	0,383972	0,3675	0,370823	0,545325	0,237621	0,102194	0,274731	0,185414	0,654944	0,89382	0,429131	0,302005	0,833365	0,624244	0
0,963824	0,965718	0,938636	0,031504	0,639711	0,612764	0,264515	0,312456	0,121349	0,258389	0,517186	0,349766	0,414118	0,847039	0,888632	0,652616	1
0,41891	0,912039	0,863206	0,930109	0,357576	0,853507	0,155757	0,317016	0,230448	0,906537	0,108466	0,01025	0,562442	0,591233	0,671954	0,842177	1
0,363344	0,45853	0,539821	0,269633	0,745519	0,770869	0,737603	0,809859	0,93627	0,028717	0,305158	0,755108	0,847037	0,879471	0,871201	0,518812	1
0,59558	0,745325	0,608003	0,283004	0,252178	0,516107	0,150763	0,509484	0,07936	0,64538	0,122168	0,925904	0,67399	0,053964	0,302224	0,818313	0
0,829403	0,828097	0,427027	0,3605	0,579184	0,451194	0,246705	0,816261	0,423267	0,665272	0,05184	0,861557	0,037813	0,13014	0,69131	0,957369	1
0,804481	0,014672	0,872773	0,078888	0,496554	0,144571	0,573789	0,776697	0,501721	0,098619	0,73494	0,267909	0,978779	0,782114	0,395892	0,898556	1
0,864798	0,065708	0,323928	0,457312	0,700881	0,114974	0,984963	0,573998	0,881961	0,234417	0,426514	0,421874	0,889889	0,277131	0,641302	0,94312	1
0,091429	0,568291	0,064164	0,04113	0,624047	0,713734	0,430694	0,276466	0,768249	0,253009	0,641041	0,506023	0,807896	0,783803	0,712501	0,689894	0
0,241578	0,097155	0,906581	0,882068	0,284653	0,749823	0,452553	0,265698	0,500521	0,870991	0,79763	0,36727	0,783237	0,254682	0,338346	0,296573	1
0,078849	0,964618	0,517738	0,317768	0,639098	0,364661	0,74245	0,163562	0,079951	0,230271	0,043734	0,311867	0,441638	0,597151	0,089603	0,679228	0
0,488002	0,868901	0,754561	0,581886	0,362326	0,494026	0,238378	0,285406	0,198294	0,543181	0,414573	0,607136	0,48508	0,986383	0,975116	0,198425	1
0,807409	0,069966	0,406408	0,403076	0,760063	0,274737	0,541576	0,037864	0,38601	0,249463	0,210926	0,550542	0,805987	0,712892	0,176813	0,985534	0
0,51513	0,901377	0,707216	0,452277	0,845251	0,673901	0,302939	0,033033	0,893014	0,435069	0,723393	0,705196	0,473254	0,340938	0,384998	0,840502	1
0,556862	0,667654	0,233042	0,544111	0,80591	0,756664	0,690132	0,517381	0,407574	0,112282	0,308461	0,562495	0,295423	0,864831	0,848528	0,26449	1
0,550466	0,43739	0,621109	0,781719	0,083888	0,338202	0,031854	0,839941	0,493642	0,792892	0,940874	0,362768	0,364197	0,273355	0,012802	0,697807	0
0,841198	0,847673	0,995713	0,64812	0,543662	0,191925	0,426753	0,440423	0,396732	0,51044	0,134976	0,930236	0,651458	0,924483	0,086631	0,572095	1
0,130992	0,272333	0,411469	0,506524	0,5092	0,965929	0,424022	0,937566	0,752351	0,793313	0,560118	0,006207	0,7698	0,370185	0,159572	0,806324	1
0,55729	0,820508	0,483818	0,708117	0,886767	0,883992	0,013977	0,282531	0,151709	0,068019	0,439966	0,123961	0,329834	0,838284	0,00788	0,20453	0
0,354511	0,78206	0,866683	0,120617	0,876827	0,216669	0,327004	0,805947	0,888909	0,172672	0,736025	0,832914	0,341833	0,443868	0,158662	0,161625	1

Şekil 1. Normalizasyon Sonrası Verilerden Bir Ekran Görüntüsü

B. Veri Hazırlığı ve Filtreleme

Bu bölümde, veri setinin eğitime nasıl uygun hale getirildiği, veri ön işleme adımlarında hangi yöntemlerin nasıl uygulandığı ve ulaşmak istenilen sonuca gelmekte nasıl yollar izlendiği hakkında bilgi verilecektir.

1. Temel Bileşenler Analizi (Principal Component Analysis - PCA)

Temel Bileşenler Analizi (PCA) Aslen Karhunen–Loève dönüşümü (KLT) olarak bilinir (Hotelling, 1933). PCA, çok sayıda değişkenin daha az sayıda temel değişken tarafından temsil edilmesini açıklar. Belirli bir p-boyutlu veri kümesi X için, $1, m, n$ asal eksenler $T1, T2, T3, ..., Tm$ olmak üzere burada $1 \le m \le p$, üzerinde tutulan varyansın yansıtılan alanda maksimum olduğu ortonormal eksenlerdir. Daha sonra, matris Tcan, örnek kovaryans matrisinin önde gelen özvektörleri tarafından verilebilir (Kara ve Direngali, 2007)

$$C = \frac{1}{N}\sum_{i=1}^{N}(x_i - a)^T(x_i - a) \qquad \textbf{3-1}$$

Burada $xi \in X$, a örnek ortalama, N ise örnek sayısıdır. Öyle ki,

$$CT_i = l_i T_i, i \in 1, 2, ..., m \qquad \textbf{3-2}$$

Burada l_i, C'nin i'ninci en büyük öz değerleridir. Verilen bir gözlem vektörünün

$$y = \left[y_1, y_2, \ldots, y_m \right] = \left[T_1^T x, T_2^T x, \ldots, T_m^T x \right] = T^T x \qquad \textbf{3-3}$$

$x \in X$ 'in ana bileşenleri aşağıdakiler tarafından verilir:

X 'in ana bileşenleri, öngörülen alanda birbirleriyle ilişkilendirilir. Bu ana bileşenler eğitim seti olarak alınıyorsa, y boyutu x boyutundan daha küçük olduğundan bilgisayarın zamanından ve belleğinden tasarruf etmek mümkün olacaktır.

2. Normalizasyon (Normalization, N)

Normalizasyonun amacı, Öklid normu matrisiyle elde edilebilecek 0 dan 1 e kadar olan tasarım değerlerine karakter kazandırmaktır. Öklid norm matrisi, gerekli temel standartlardan biridir. Matris elemanlarının karelerinin toplamının karekökü olarak tanımlanır.

$$\left\| A \right\|_E = \sqrt{\sum_{i=1}^{n} \sum_{j1}^{m} a_{ij}^2} \qquad \textbf{3-4}$$

Burada a_{ij}, matrisin i-inci satır ve j-inci sütundaki elemanı, n satır sayısı, m ise sütun sayısıdır. Normalizasyon sonucunda veri kümesindeki sayısal sütunların değerleri, değerler aralığını bozmadan ortak bir ölçeğe yerleştirilmiş olur (Bkz. Şekil 1)

3. Düşük Varyasyon Filtresi (Low Variance Filter – LVF)

Veri setinde, arasında küçük değişiklikler olan veri sütunları çok az bilgi taşır. Bunları filtrelemek için düşük varyans filtresi uygulanır. Bu filtre, kullanıcının tanımladığı bir eşiğin altında kalan kolonları eğitimin dışında tutmaya yarar. Varyansı düşük sütunlar bazı öğrenme algoritmalarının özellikle de mesafeye dayalı olan öğrenme algoritmalarını sonucunu negatif yönde etkileyebileceği için filtrelenmesi gerekir. Varyansın yalnızca sayısal sütunlar için hesaplanabileceğine dikkat etmek gerekir, yani bu boyutluluk azaltma yöntemi yalnızca sayısal sütunlar için geçerlidir. Ayrıca, varyans değerinin sütun sayısal aralığına bağlı olduğunu

unutmamak gerekir. Bu nedenle, varyans değerlerini sütun alanı aralığından bağımsız hale getirmek için varyanslarını hesaplamadan önce veri sütunu aralıkları normalleştirilmelidir. İlk önce bir normalizer düğümü tüm sütun aralıklarını [0, 1] olarak normalleştirir; daha sonra, düşük varyanslı filtre düğümü sütun varyansını hesaplar ve sütunları belirlenmiş bir eşiğin altında bir varyansa göre filtreler; son olarak, kalan tüm sütunlar orijinal sayısal aralıklarına geri dönecek şekilde normalleştirilir.

4. Yüksek Korelâsyon Filtresi (High Correlation Filter – HCF)

Çok yakın eğilimlere sahip veri sütunlarının benzer bilgiler taşıması olasıdır. Bu durumda, yalnızca biri makine öğrenme modelini beslemek için yeterli olacaktır. Burada nümerik sütunlar arasında ve nominal sütunlar arasında sırasıyla Pearson'un Moment Katsayısı (Url-14) ve Pearson ki kare değeri (Bolboaca vd. , 2011) olan korelâsyon katsayısı hesaplanır. Korelâsyon katsayısı eşikten daha yüksek olan sütun çiftleri yalnızca bir taneye düşürülür. Korelâsyonun ölçeğe duyarlı olması nedeniyle anlamlı bir korelâsyon karşılaştırması için sütun normalizasyonu gereklidir.

C. Metodoloji

Seçilen makine öğrenmesi algoritmalarından bahsedilmekte ve bahsi geçen algoritmalar kullanılırken elde edilen grafikler, tablolar ve şekiller gösterilmektedir. Bütün matematiksel hesaplama ve sonuç çıkarımı için Python, C ve C++ dillerinden, şekillerin ortaya çıkarılması için de RapidMiner (RapidMiner Studio 9.4.001 (rev: db3dae, platform:WIN64) programından faydalanılmıştır. Algoritmalar uygulanırken önce veri kümesi yüklenmiş ve ön işlemeler gerçekleştirilmiştir. Tüm etiketli veri noktaları ve modelin daha sonra uygulanması gereken etiketlenmemiş olanları ortaya çıkartılmıştır. Bu veri setinden bir eğitim ve doğrulama seti oluşturulmuştur ki bu doğrulama (validation) seti, sağlam bir çoklu performans hesabında kullanılacaktır. Bu yapılan işlemlere veri ön işleme adı verilir. Bu makine öğrenmesi yöntemlerini karşılaştırmak için gerekli olan veri setimizde sırasıyla normalizasyon, düşük varyans filtresi, yüksek korelâsyon filtresi ve temel bileşenler analizi kullanılmıştır. Ardından model eğitimi ve otomatik hiperparametre ayarı (parametre optimizasyonu) gerçekleştirilmiştir ki buna özellik mühendisliği ya da modelleme denmektedir. Aynı ön işleme ve özellikleri kullanarak doğrulama verileri (hedef değer bilinmektedir)

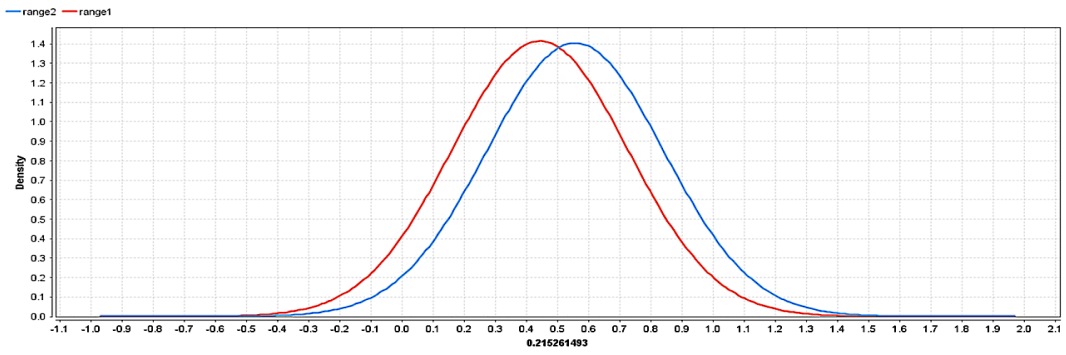
dönüştürülmüştür ve ardından aynı ön işleme ve özellikleri kullanarak puanlama verileri (hedef değer bilinmemektedir) dönüştürülmüştür. Modelin geçerliliğine ve puanlama verisine yönelik puanlama verilerini uygulanmıştır ve tahminlerle birlikte modele özgü ağırlıklar hesaplanmıştır. Bunların ardından daha küçük çalışma sürelerine sahip çapraz onaylamaya benzer performans tahmini kalitesi sağlayan sağlam kestirim ile çoklu bir onaylama seti doğrulaması gerçekleştirilmiştir ve ardından model simülatörü oluşturulmuştur. Daha sonra birleşik eğitim ve onaylama veri setlerinde aynı parametrelere sahip bir model eğitilerek son bir üretim modeli oluşturulmuştur. Bunlardan sonra çalışma süreleri toplanmıştır ve çıkan sonuçlar, sonuç bağlantı noktalarına gönderilmiştir.

1. Naif Bayes (Naîve Bayes – NB)

Naif Bayes (NB) sınıflandırıcısı, kolay ve hızlı bir şekilde tanımlanır, çünkü sınıflandırma için parametreleri (değişken ve araç değişkenleri) tahmin etmek için küçük bir eğitim verisi gerektirir (Keramati ve Yousefi, 2011). NB sürecinin arkasındaki ana fikir, veri kümelerinin özniteliklerini bağımsız olarak araştırmaktır (Kim, 2009; Hou ve ark. , 2010; Murphy, 2006) ve çok büyük veri setleri için modeli genişletmek kolaydır. Bu sınıflandırıcı metodu, bir doküman sınıflandırması algoritmasını öğrenir ve Bayes kuralının basit kullanımına dayanır.

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)} \qquad \textbf{3-5}$$

Burada c bir sınıf, d bir belge, P (c) bir olasılık, P (d) belgenin olasılığı, P (d | c) verilen d belgesi için sınıfın şartlı olasılığı, P (c | d) ise d belgesinin c sınıfına ait koşullu olasılığıdır. Naif Bayes modelinden bir parametrenin görüntüsü Şekil 2'de gösterilmiştir



Şekil 2. Naif Bayes Algoritması Ağırlıklarından Birinin Yoğunluk Grafiği

Naif Bayes sınıflandırıcısında kullanılan ağırlıklar Çizelge1'deki gibidir.

Çizelge 1. Naif bayes algoritması ağırlıkları

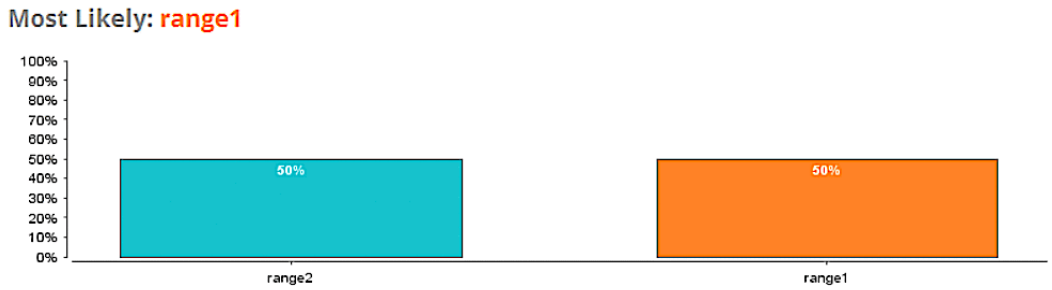
Nitelik	**Ağırlık**
0.329169068	0,2
0.04058144	0,1
0.340776522	0,1
0.304858309	0,1
0.350843754	0,1
0.950809496	0,1
0.090783852	0,1
0.218576167	0,1
0.591770924	0,1
0.09923022	0,1
0.804403656	0,1
0.215261493	0,1
0.086314274	0,1
0.788656772	0,1
0.937680009	0,1
0.495146927	0

Simülasyonda çıkması muhtemel sonuç Şekil 3'de gösterilmiştir. Burada, veri düzgünleştirme için gruplama (Binning) yöntemiyle ilgilenilmiştir. Bu yöntemde veriler önce sıralanır ve ardından sıralanan değerler birkaç kova veya kutuya dağıtılır. Binicilik yöntemleri değerlerin bulunduğu lokasyona başvurduğundan, lokal düzeltme işlemi gerçekleştirir.

En basit gruplama yaklaşımı, değişkenin aralığını k eşit genişlik aralıklarına bölmektir. Aralık genişliği basitçe değişkenin k ile bölünen değişkenin $[A, B]$ aralığıdır,

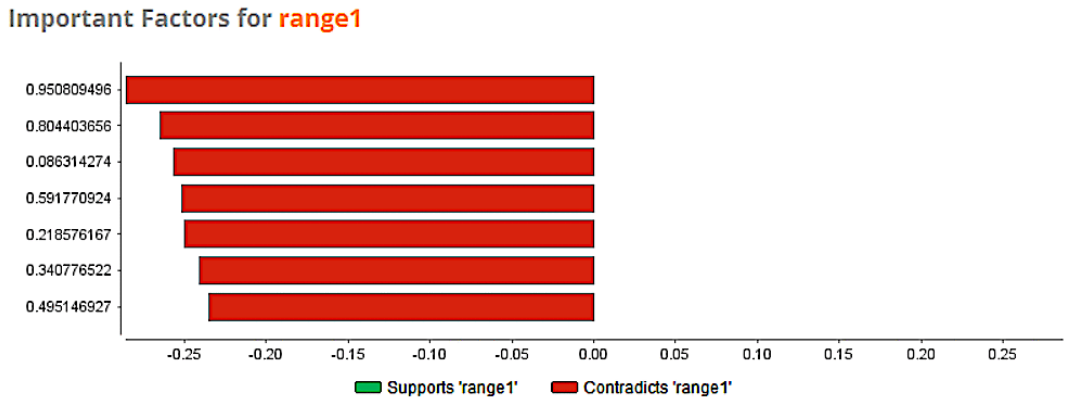
$$w = (B - A) \ / \ k \qquad \textbf{3-6}$$

Böylece, aralık aralığı $\left[A + (i-1)\, w,\ A + iw\right]$ olacaktır, burada $i = 1, 2, 3\ldots ..k$ olarak kabul edilir. Bu yönteme de eşit genişlik ya da mesafe gruplaması denir. Bu gruplamalardan sonra simülasyon, muhtemel gruplamalardan hangisinin daha iyi sonuç vereceğini ortaya çıkartacaktır. Bu sonuç, naif bayes algoritması için şekil 3'de gösterilmiştir.



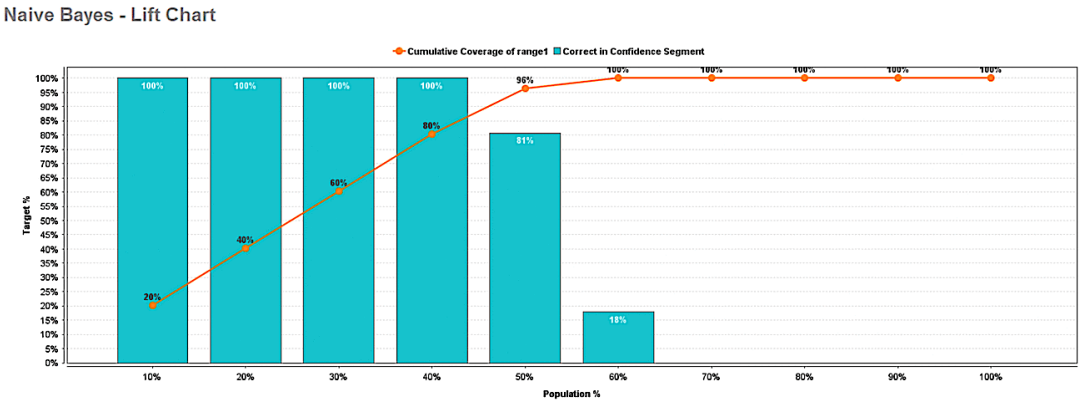
Şekil 3. Naif Bayes Algoritması Simülasyon Gruplaması Sonucu

Bu muhtemel sonuç için önemli faktörler Şekil 4'deki gibidir.



Şekil 4. Naif Bayes Algoritması Uzaklık-1 için Önemli Faktörler

Naif Bayes Lift Chart (Yükseltme Grafiği) , bir model kullanarak (Karar Ağacı / Lojistik Regresyon gibi) rastgele tahmin tahminlerini herhangi bir makine öğrenme modeli olmadan ölçmek için kullanılır. Rastgele tahminden kaynaklanan tahminin bu şekilde iyileştirilmesine yükseltme (lift) denir. Naif Bayes algoritmasının yükseltme algoritması Şekil 5'de gösterilmiştir.



Şekil 5. Naif Bayes Algoritması Yükseltme Grafiği

Naif bayes algoritması, hesaplamalı verimlilik, düşük varyans, artan öğrenme, gürültüye dayanıklılık ve eksik değerlerde sağlamlık açısından iyi performans göstermektedir (Sammut ve Webb, 2017).

2. Genelleştirilmiş Doğrusal Model (Generalized Linear Model – GLM)

$$E[Y] = \mu = g^{-1}(Xa) \qquad \textbf{3-7}$$

Doğrusal bir belirleyici olan $X\alpha$, bilinmeyen parametre olan a'ların doğrusal bir birleşimidir ve g, link işlevi olarak adlandırılır. Link fonksiyonu, bağımlı değişkenin ortalamasının bir dönüşümüdür, öyle ki bu dönüştürülmüş değişken, regresyon parametrelerinin doğrusal bir işlevidir.

Yanıt değişkeninin varyans'ı tipik olarak aşağıdaki gibi ortalamanın bir fonksiyonudur:

$$\text{var}[Y] = V(\mu) = V\left[g^{-1}(Xa)\right] \qquad \textbf{3-8}$$

Bilinmeyen parametre olan a genellikle, analitik olarak çözülemeyen bir denklem sisteminden maksimum olasılık, Yarı-maksimum olabilirlik veya Bayesian tahmincisi gibi teknikler kullanılarak tahmin edilir.

Genelleştirilmiş doğrusal model, aşağıdaki gibi üç unsurdan oluşur

1) Üstel bir dağılım ailesinden f bir dağıtım fonksiyonu

2) Doğrusal bir tahmin sağlayıcı $\eta = X\alpha$

3) $E[Y] = \mu = g - 1(\eta)$ gibi bir link fonksiyonu

Genelleştirilmiş doğrusal modelde kullanılan model Çizelge 2'deki gibidir.

Çizelge 2 – Genelleştirilmiş doğrusal model'in modeli

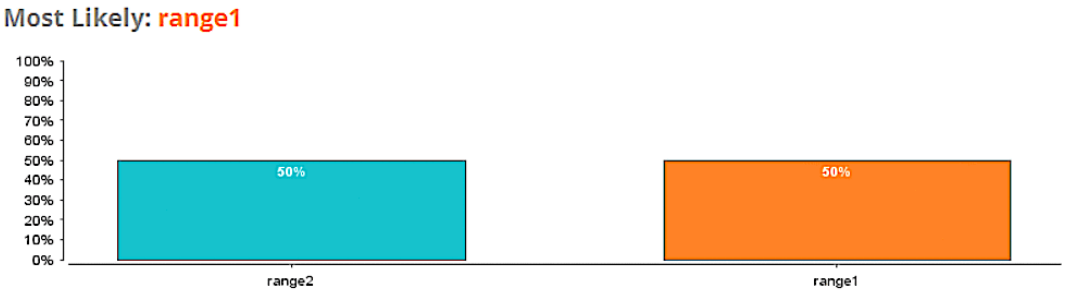
Attribute	Coefficient	Std. Coefficient
0.215261493	-14.338	-4.140
0.329169068	-14.347	-4.147
0.788656772	-14.366	-4.142
0.950809496	-14.321	-4.128
0.591770924	-14.444	-4.163
0.86314274	-14.385	-4.155
0.804403656	-14.394	-4.145
0.350843754	-14.412	-4.167
0.9923022	-14.483	-4.193
0.4058144	-14.246	-4.105
0.218576167	-14.329	-4.137
0.90783852	-14.457	-4.189

Genelleştirilmiş doğrusal modelde kullanılan ağırlıklar Çizelge 3'de verilmiştir.

Çizelge 3 – Genelleştirilmiş doğrusal model ağırlıkları

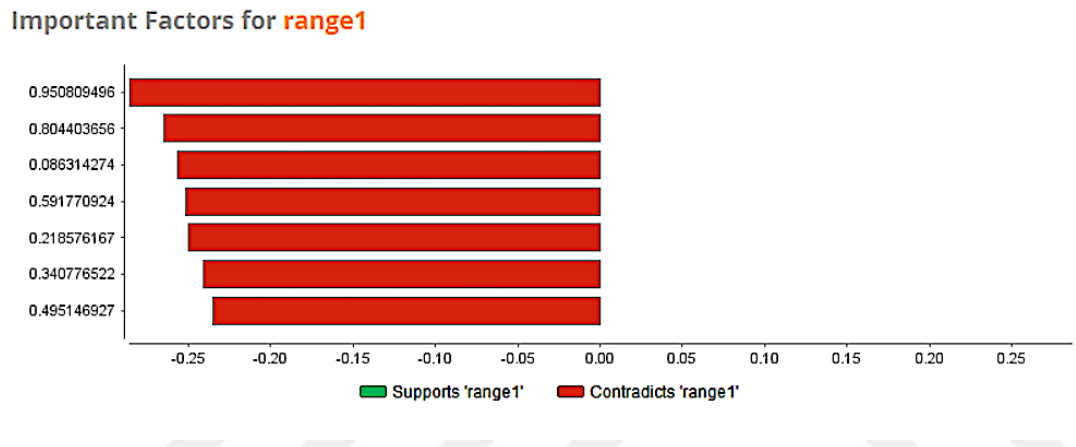
Nitelik	Ağırlık
0.329169068	0,2
0.04058144	0,1
0.304858309	0,1
0.340776522	0,1
0.350843754	0,1
0.950809496	0,1
0.090783852	0,1
0.218576167	0,1
0.591770924	0,1
0.09923022	0,1
0.215261493	0,1
0.804403656	0,1
0.086314274	0,1
0.937680009	0,1
0.788656772	0,1
0.495146927	0

Eşit genişlik gruplamasından sonra simülasyonda ortaya çıkan gruplama sonucu şekil 6'daki gibidir.



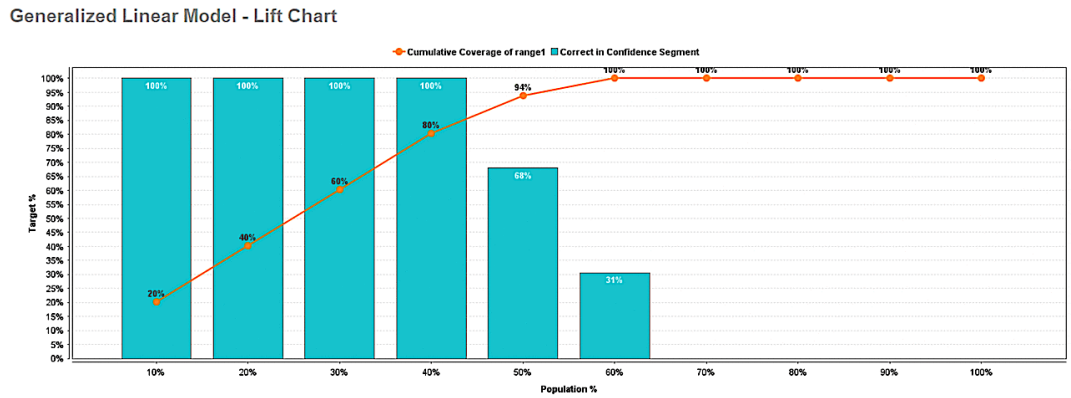
Şekil 6. Genelleştirilmiş Doğrusal Model Algoritması için Simülasyon Sonucu

Bu muhtemel sonuç için önemli faktörler Şekil 7'de gösterilmiştir



Şekil 7. Genelleştirilmiş Doğrusal Model Uzaklık-1 için Önemli Faktörler

Genelleştirilmiş doğrusal model algoritması için yükseltme grafiği Şekil 8'deki gibidir.



Şekil 8. Genelleştirilmiş Doğrusal Model Yükseltme Grafiği

Bu modelin iyi yanları, normal bir dağılıma sahip olması için Y yanıtının (cevap değişkeninin olasılık dağılımı) değişmesine gerek olmaması, bağlantı seçimi rastgele bileşen seçiminden ayrı olduğundan modelleme konusunda daha fazla esnekliğe

sahip olması, eğer bağlantı ilave etkiler yaratıyorsa sabit varyansa ihtiyaç olmaması, modellerin maksimum olabilirlik tahmini ile donatılmış yani tahmin edici özelliklerin optimum çalışması, log-linear ve lojistik regresyon için kullanılan tüm özellik çıkarım araçları ve model kontrollerinin bu algoritmada da çalışıyor olması (örneğin ward ve olabilirlik oranı testleri, sapma, artıklar, güven aralıkları, aşırı dağılma) ayrıca yazılımlarda bu modelleri yakalamak için prosedürler olmasıdır (Örneğin R'daki SAS ya da glm() PROC GENMOD)

3. Lojistik Regresyon

Genel olarak, veri setindeki her veri noktası için şablon görevi görebilen $\overline{X}$ = $x1, x2, ..., xn$ özelliklerinin bir vektörüyle başlar. Hayatta kalmayı öngören ikili bir Y sınıflandırıcısı oluşturulmalıdır. Bu yapım temel olarak eğitim veri setinin özelliklerine dayanmaktadır. Klasik regresyonda (Lin ve ark. 2008), X vector özellik vektörü denklemdeki veri noktasına uyması için kullanılır:

$$P(\overline{X}) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n \qquad \textbf{3-9}$$

Bu değer mutlaka 0 ile 1 arasında olmadığından, veri noktasına bir sınıf atamak için olasılık olarak kullanılamaz. Genel olarak, bir link fonksiyonu logit, $\overline{X}$: *to a* 'yı 0 ile 1 arasında bir değere dönüştürmek için kullanılır.

$$\mathrm{P} = \frac{e^{\beta_0+\beta_1 X_1+\beta_2 X_2+...+\beta_n X_n}}{1+e^{\beta_0+\beta_1 X_1+\beta_2 X_2+...+\beta_n X_n}} \qquad \textbf{3-10}$$

Eğitim verileri, Denklem 3-10'un katsayılarını tahmin etmek için kullanılır. Sınıf atamasına karar vermek için log olasılığı tercih edilir. İkili sınıflandırma için, her eğitim verisi noktası $\overline{X}$ i bir sınıf ataması yi (ör. survival için 0, non-survival için 1). Daha sonra olasılığını hesaplamak için denklem 3-10'daki bir veri noktası değiştirilir. Bunun logaritmik ihtimali,

$$\sum_{i=1}^{n} y_i \log p_i + (1+\gamma_i)\log(1+p_i) \qquad \textbf{3-11}$$

Denklem 3-11, verilerden katsayılar için Maksimum Olabilirlik Tahmini (Maximum Likelihood Estimation - MLE) elde etmek için sayısal olarak çözülür.

Lojistik regresyonda kullanılan model Çizelge 4'deki gibidir.

Çizelge 4 – Lojistik regresyon modeli

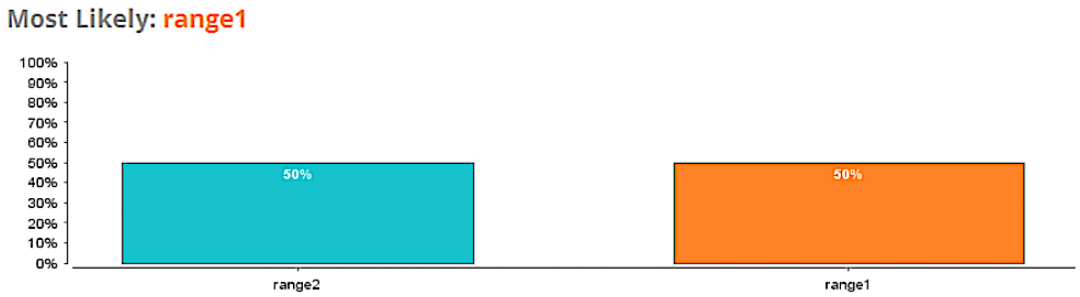
Attribute	Coefficient	Std. Coefficient
0.215261493	-14.338	-4.140
0.329169068	-14.347	-4.147
0.788656772	-14.366	-4.142
0.950809496	-14.321	-4.128
0.591770924	-14.444	-4.163
0.86314274	-14.385	-4.155
0.804403656	-14.394	-4.145
0.350843754	-14.412	-4.167
0.9923022	-14.483	-4.193
0.4058144	-14.246	-4.105
0.218576167	-14.329	-4.137
0.90783852	-14.457	-4.189

Bu modelde kullanılan ağırlıklar Çizelge 5'deki gibidir

Çizelge 5 – Lojistik regresyon'da kullanılan ağırlıklar

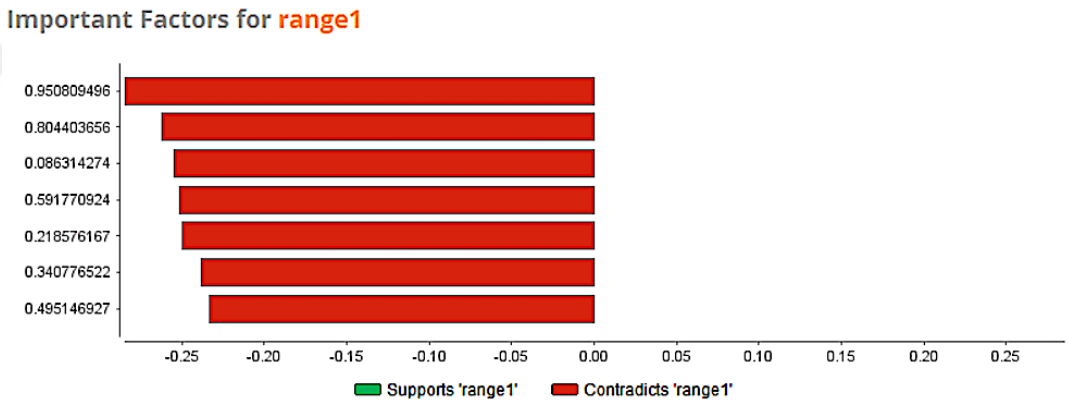
Nitelik	Ağırlık
0.329169068	0,2
0.04058144	0,1
0.304858309	0,1
0.340776522	0,1
0.350843754	0,1
0.950809496	0,1
0.090783852	0,1
0.218576167	0,1
0.591770924	0,1
0.09923022	0,1
0.215261493	0,1
0.804403656	0,1
0.086314274	0,1
0.937680009	0,1
0.788656772	0,1
0.495146927	0

Simülasyon gruplaması sonucu Şekil 9'da gösterilmiştir.



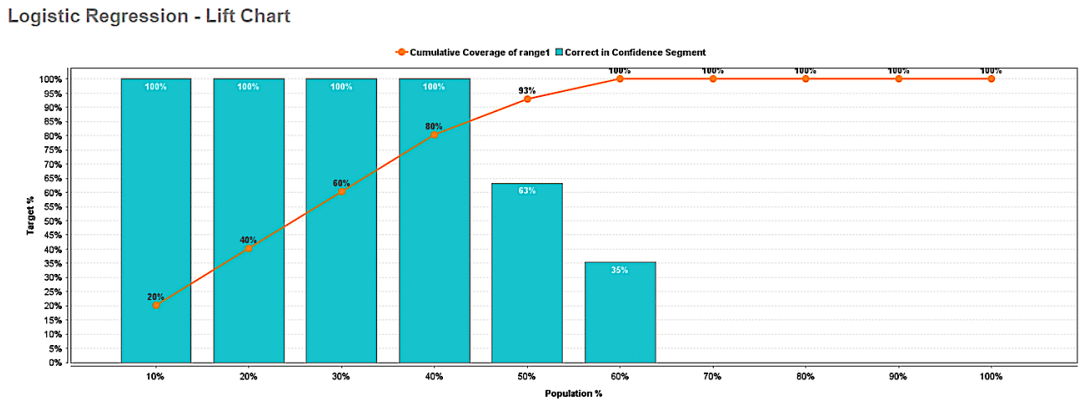
Şekil 9. Lojistik Regresyon Algoritması Simülasyon Gruplaması Sonucu

Bu algoritma için önemli faktörler Şekil 10'da gösterilmiştir.



Şekil 10. Lojistik Regresyon Algoritması Uzaklık-1 için Önemli Faktörler

Bu algoritma için yükseltme grafiği ise Şekil 11'deki gibidir.



Şekil 11. Lojistik Regresyon Yükseltme Grafiği

Lojistik Regresyon, yaygın bir şekilde kullanılan bir tekniktir çünkü verimli çalışır ve çok fazla hesaplama kaynağı gerektirmez. Kolay yorumlanabilirliği, girdi özelliklerinin ölçeklendirilmesini gerektirmemesi, herhangi bir ayar gerektirmemesi,

düzenlenmesinin kolay olması ve iyi kalibre edilmiş tahmin edilen olasılıklar vermesi tercih sebeplerinden olmuştur. Lojistik regresyon, çıktı değişkeniyle ilgili olmayan nitelikleri ve birbirine çok benzeyen (ilişkili) nitelikler kaldırıldığında daha iyi çalışır. Bu nedenle, Özellik Mühendisliği, Lojistik ve Doğrusal Regresyon performansı açısından önemli bir rol oynamaktadır. Sadeliği ve nispeten kolay ve hızlı bir şekilde uygulanabilmesi nedeniyle, Lojistik Regresyon, diğer daha karmaşık Algoritmaların performansını ölçmek için kullanılabilecek iyi bir temeldir (Url-21).

4. Karar Ağaçları

1960'lı yıllardan beri, karar ağacı yöntemi, sınıflandırma, tahmin ve kural olarak çeşitli alanlarda yaygın olarak uygulanmaktadır. Geleneksel karar ağacının, $X > Ti$ gibi bir şartı ortaya koyarken bir eşik değeri yapması gibi birçok dezavantajı vardır; bu nedenle, eğer resmi değişken değer eşik değerin üstünde değişirse, sonuç süreksizlik olarak adlandırılan akut değişecektir. Sonuç, eşik değerine karşı çok hassastır ve bilgi anlayışı çok iyi değildir. Bahsedilen dezavantajların çözümleri, bulanık dağılımın hedef değişkene eksprese edilmesidir. Bulanık karar ağacı anlaşılırlığı artırmak ve karar sonuçlarının sürekliliğini sağlamak ve sınıflandırmak için benimsenebilir.

$E = D1 \times D2 \times \ldots Dn$ boyutunun n olduğu sonlu vektör alanı olarak Dn ve Dj sonlu dağılım işaretleme toplanışıdır, E'deki $e = e = (v1, v2, \ldots vn)$ olarak adlandırılan öğeye örnek $vj \in Dj,\ j = 1, 2, \ldots, n$ denir. PE *ve* NE, pozitif ve negatif örnek toplayıcılar olarak adlandırılan E'nin örnek toplayıcılarıdır. Şimdi, PE *ve* NE 'nin p ve n olduğunu, ID3 temelli, doğru karar ağacının sınıflandırılma olasılığının, E'deki pozitif ve negatif örnekleme ihtimaliyle tutarlı olduğunu ve bire yönelik doğru sınıflandırma kestirimi yapmanın bilgi bitiyle tutarlı olduğunu gösterir. Örnek aşağıdaki gibidir:

$$I(p,n) = \frac{p}{p+n}\log_2\frac{p}{p+n}\frac{n}{p+n}\log_2\frac{n}{p+n} \qquad \textbf{3-12}$$

E, karar ağacının kökü olarak E'yi v alt sınıfına bölen v değerine sahip A niteliği yapsa, bu arada Ei'de pi pozitif örnekler ve ni negatif örnekler olduğunu varsayalım, Ei'nin beklenen bilgisi $I\ (pi,\ ni)$ ve A'ya dayanan bekleme bilgisi:

$$E(A) = \sum_{i=1}^{n} \frac{p_i + n_i}{p+n} I(p_i, n_i) \qquad \textbf{3-13}$$

Yani A'nin kazanımı,

$$gain(A) = I(p,n) - E(A) \qquad \textbf{3-14}$$

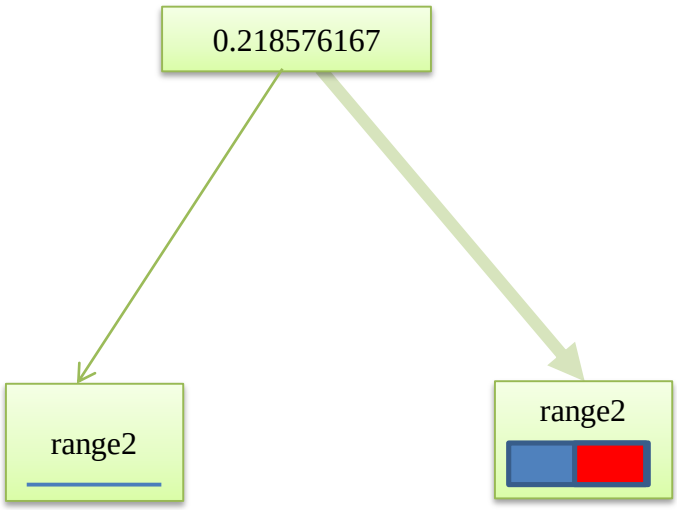
ID3 algoritması, A niteliğini, (A) kazancını en büyük hale getirebilecek, sıradan bir özellik olarak seçer ve bu yöntem, daha hızlı sınıflandırma hızını elde etmek için karar ağacı ortalama derinliğini daha sığ hale getirebilir. Ancak uygulama, bu standardın daha fazla değer alabilecek niteliklere sahip olmanın kolay olduğunu kanıtlamıştır. Artım oranı;

$$gain - ratio(A) = gain(A) / IV(A) \qquad \textbf{3-15}$$

Olmak üzere,

$$IV(A) = \sum_{i=1}^{n} \frac{p_i + n_i}{p+n} I(p_i, n_i) \qquad \textbf{3-16}$$

A özelliğinin değerinin oluşturulması hakkında bilgi ölçümüdür. Kullanılan karar ağacı modeli Şekil 12'deki gibidir.



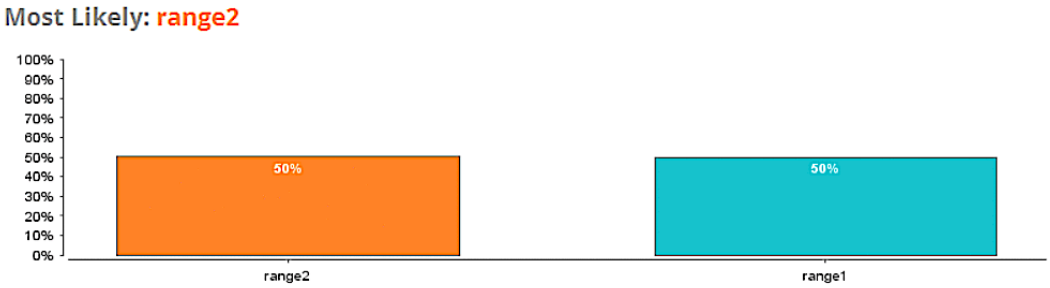
Şekil 12. Karar Ağacı Modeli

Karar ağacı algoritmasındaki ağırlıklar Çizelge 6'da gösterilmiştir.

Çizelge 6 – Karar ağacı algoritmasında kullanılan ağırlıklar

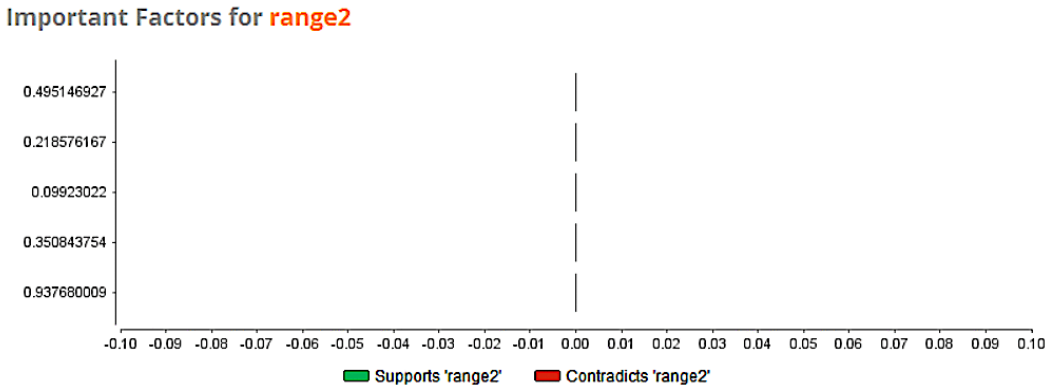
Nitelik	**Ağırlık**
0.218576167	0,1
0.937680009	0
0.495146927	0
0.09923022	0
0.350843754	0

Simülasyon gruplaması sonucu Şekil 13'de gösterilmiştir.



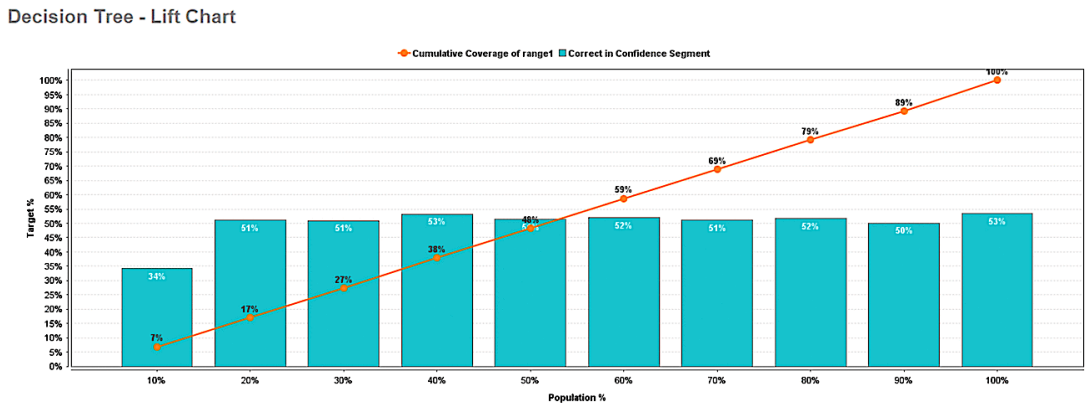
Şekil 13. Karar Ağacı Algoritması Simülasyon gruplaması sonucu

Bu muhtemel sonuç için önemli faktörler Şekil 14'de gösterilmiştir.



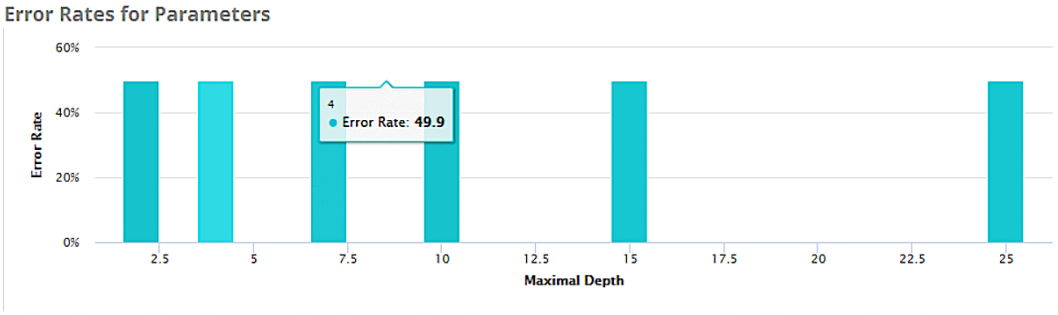
Şekil 14. Karar Ağacı Algoritması için önemli faktörler

Bu algoritma için Yükseltme Grafiği Şekil 15'deki gibidir.



Şekil 15. Karar Ağacı Algoritması Yükseltme Grafiği

Karar ağacı algoritması uygulanması sırasında ortaya çıkan optimal parametrelerden birkaçı Şekil 16'da gösterilmiştir.



Şekil 16. Karar Ağacının Optimum Parametreleri

Karar ağaçlarının anlaması, yorumlaması, görselleştirmesi basit olarak sayılmaktadır. Bu ağaçlar örtülü olarak değişken tarama veya özellik seçimi yaparlar. Hem sayısal hem de kategorik verileri yönetebilir ve çok çıkışlı sorunları da çözebilirler. Karar ağaçları, verilerin hazırlanması aşamasında daha az çaba ister. Ayrıca, parametreler arasındaki doğrusal olmayan ilişkiler karar ağaçları performansını etkilemez.

Karar ağacı eğitimleri, verileri iyi genelleştirmeyen karmaşık sistemler oluşturabilirler. Buna aşırı uyma denir (over-fitting). Karar ağaçları kararsız olabilir, çünkü verilerdeki küçük farklılıklar tamamen farklı bir ağacın üretilmesine neden olabilir. Buna torbalama ve güçlendirme gibi yöntemlerle indirilmesi gereken varyans denir. Açgözlü algoritmalar globâl olarak en uygun karar ağacını geri getirmeyi garanti edemez. Bu özelliklerin ve örneklerin değiştirmeyle rastgele örneklendiği birden fazla ağacı eğiterek hafifletebilir. Karar ağacı öğrenenler, bazı sınıfların hâkim olması durumunda önyargılı ağaçlar oluşturur. Bu nedenle karar ağacına uymadan önce, ayarlanan verilerin dengelenmesi önerilir.

5. Rastgele Orman (Random Forest – RF)

Topluluk ya da grup öğrenmesi, bireysel öğrenicileri eğiterek ve tahminlerini kaynaştırarak sınıflandırmayı öğrenen koleksiyonlar veya sınıflandırıcılar topluluğu üreten algoritmaları ifade eder. Bir ağaç topluluğu yetiştirmek ve en popüler sınıfa oy vermelerini sağlamak, sınıflandırma doğruluğunda iyi bir gelişme sağlamıştır. Genellikle, topluluktaki her ağacın büyümesini kontrol eden rastgele vektörler oluşturulur. Topluluk öğrenme yöntemleri iki ana gruba ayrılabilir: torbalama ve güçlendirme. Torbalamada, ardışık ağaçların önceki ağaçlara bağlı olmadığı modeller paraleldir. Her ağaç bağımsız olarak veri kümesinin önyükleme örneği kullanılarak oluşturulur. Çoğunluk oyu öngörüyü belirler. Güçlendirmede, ardışık ağaçların önceki model tarafından kötü bir şekilde öngörülen bu gözlemlere ek ağırlık atadığı yerlerde sıralı olarak modeller uygundur. Ağırlıklı oylama, öngörüyü belirtir. Rastgele bir orman, torbalamaya ek bir rastgele derece derecesi ekler. Her ağaç, veri kümesinin farklı bir önyükleme örneği kullanılarak oluşturulmuş olmasına rağmen, ağaçların sınıflandırıldığı yöntem geliştirilmiştir. Rastgele bir orman tahmincisi, bireysel bir sınıflandırma ağacı tahmincisi grubudur. Her gözlem için, her bir ağaç bir sınıfa oy verirken, orman çok sayıda oy alan sınıfı tahmin eder. Kullanıcı, her bir düğümdeki en iyi bölünme için aranacak rastgele seçilen değişkenlerin (mtry) sayısını belirtmek zorundadır.

Bir düğüm standart ağaçlardaki tüm değişkenler arasında en iyi bölünme kullanılarak bölünürken, rastgele bir ormanda düğüm, o düğümde rastgele seçilen bir belirteç alt kümesi arasında en iyiyi kullanarak bölünür. Mümkün olan en büyük ağaç yetiştirilir ve budanmaz. Ormandaki her ağacın kök düğümü, eğitim seti olarak orijinal verilerden bir önyükleme örneği içerir. Eğitim setinde bulunmayan gözlemlere "çanta dışı" gözlemler denir. Tek bir ağacın açılmamış olması nedeniyle, terminal düğümleri sadece az sayıda gözlem içerebilir. Antrenman verileri her bir ağacın üzerinden geçirilir. Eğer gözlemler i ve j'nin ikisi de aynı terminal düğümünde sona ererse, i ve j arasındaki benzerlik bir artırılır. Orman inşaatının sonunda benzerlikler simetrik hale getirilir ve ağaç sayısına bölünür. Bir gözlem ve kendisi arasındaki benzerlik bir olarak belirlenmiştir. Nesneler arasındaki benzerlikler simetrik bir matris oluşturur ve her giriş birim aralığında yer alır. Breiman rastgele orman'ı şu şekilde tanımlamıştır (Breiman, 2001):

Rastgele Orman, ağaç yapılı sınıflandırıcı koleksiyonundan oluşan bir sınıflandırıcıdır.

$$\{h(X,\Theta_k),k=1,...\} \quad \textbf{3-17}$$

Olmak üzere Θ_k bağımsız olarak aynı şekilde dağıtılmış rasgele vektörlerdir ve her ağaç X girişindeki en popüler sınıf için bir oy kullanır. Sınıflandırma için Rastgele Orman algoritması şu şekilde özetlenebilir:

- Orijinal verilerden ntree bootstrap örnekleri çizilir
- Önyükleme örneklerinin her biri için, aşağıdaki değişikliklerle birlikte, önceden tanımlanmamış bir sınıflandırma ağacı büyütülür: her bir düğümde, tüm yordayıcılar arasında en iyi ayrımı seçmek yerine, yordayıcıların rasgele mtry seçilir ve bu değişkenler arasından en iyi ayrımı seçilir. Torbalama, $mtry = p$ yordayıcı sayısı elde edildiğinde elde edilen rastgele ormanın özel durumu olarak düşünülebilir.
- ntree ağaçlarının tahminlerini toplayarak yeni verileri tahmin edilir ki bunlar da çoğunluk oylar sınıflandırma için, ortalama oylar regresyon içindir

Bir ağaç rastgele orman'ın genelleme hatası, forest'teki tek tek ağaçların gücüne ve aralarındaki korelâsyon'a bağlıdır. Her düğümü bölmek için rastgele bir özellik seçimi kullanmak, AdaBoost (Freund ve Schapire, 1999:771-780) ile karşılaştırıldığında hata oranları verir. Antrenman verilerine dayanarak hata oranının bir tahmini şu şekilde elde edilebilir (Liaw ve Wiener, 2002):

- Her bir önyükleme yinelemesinde, önyükleme örneği ile büyütülen ağacı kullanarak, "torba dışı" olarak adlandırılan önyükleme örneğinde olmayan veriler tahmin edilir.
- Torba dışı tahminleri toplanır: Ortalama olarak, her bir veri noktası, zamanın % 36'sı kadarı tükenmez olacaktır, bu nedenle bu tahminler toplanır. Hata oranı hesaplanır ve hata oranının "torba dışı" tahmini olarak adlandırılır.

6. Gradyan Arttırılmış Ağaçlar (Gradient Boosted Trees – GBT)

Gradyan artırılmış ağaçlar, ağaç topluluğu modeline aittir (Friedman, 2001:1189-1232). Bir ağaçta, her bir eğitim örneği Xi, bir yaprağa sınıflandırılır. Her bir yaprak, örneklerini bir w ağırlık ağırlığı ile tahmin eder. Bir ağaç oluşturduktan sonra, GBT karşılık gelen yaprak ağırlığını ekleyerek her bir vakanın tahminini günceller. Sonra GBT bir sonraki ağaca taşınır. GBT, her yapının öngörü sınıfını verdiği karar ağacından ziyade, bir yaprağa sürekli tahminsel ağırlık veren regresyon ağacını benimser. Tüm ağaçları bitirdikten sonra, GBT tüm ağaçların tahminlerini nihai tahmin olarak toplar (Friedman J. , vd. 2000:337-407).

$$\widehat{y_i} = \sum_{t=1}^{T} f_t(X_i) \qquad \textbf{3-18}$$

T 'ninci ağaç için aşağıdaki düzenli hedefin en aza indirgenmesi gerekir

$$F^{(t)} = \sum_{i=1}^{N} l(y_i, y_i^{(t)}) + \Omega(f_t) = \sum_{i=1}^{N} l(y_i, y_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \qquad \textbf{3-19}$$

Burada l, tahmin ve verilen lojistik kayıp gibi hedefe verilen zararı hesaplayan bir kayıp fonksiyonudur. Ω , aşırı yerleştirmeyi önlemek için ceza işlevini gösterir. LogitBoost (Friedman J. , vd. 2000:337-407), F(t) 'yi ikinci dereceden yaklaştırma ile genişletir. gi ve hi, l'nin birinci ve ikinci dereceden gradyanlarıdır.

$$F^{(t)} \approx \sum_{i=1}^{N} \left[l(y_i, y_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^{\,2}(x_i) \right] + \Omega(f_t) \qquad \textbf{3-20}$$

Optimum çözümün bulunması için mümkün olan her ağaç yapısı numaralandırılabilir. Bununla birlikte, bu şema pratikte kullanışsızdır. Bu nedenle, genellikle ağaç düğümlerinin başarılı bir şekilde bölünmesi için Şekil 17'de görülen açgözlü bir algoritma (Jiang J., vd. 2017) kullanılır.

m: # features, n: # instances, k : # split candidates
for *m= 1* to *m* **do**
    generate k split candidates sm = {sm1,sm2,…smk}
end for
for *m=1* to *m* **do**
    loop n instances to generate gradient histogram with k bins
    $g_{mk} = \sum gi \; where \; Smk - 1 < xim < smk$
    $h_{mk} = \sum hi \; where \; Smk - 1 < xim < smk$
end for

$gain_{max} = 0$, $g = \sum_{i=1}^{N} gi$, $h = \sum_{i=1}^{N} hi$
for *m = 1* to *m* **do**
    $gl = 0$, $hl = 0$
    for *k=1* to *k* **do**
        $gl = gl + gmk$, $hl = hl + hmk$
        $gr = g - gl$, $hr = h - hl$
        $gainmax = max(gainmax, \frac{G^2 l}{Hl+\sigma} + \frac{G^2 R}{Hr+\sigma} + \frac{G^2}{H+\sigma})$
    end for
end for

Şekil 17. Gradyan Arttırılmış Ağaçlar için Aç Gözlü Algoritma

Her özellik için K adet bölünmüş adaylar göz önüne alındığında, gradyan istatistiklerini özetleyen bir gradyan histogramı oluşturmak için düğümün tüm örnekleri döngü içine alınır (satır 4-8). Tüm özelliklerin gradyan istatistiklerini içeren gradyan histogramı oluşturulduktan sonra, tüm histogram kutuları numaralandırılır ve maksimum objektif kazanç sağlayan parça bulunur (satır 10-17).

$$Gain = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \qquad \textbf{3-21}$$

Burada I_L ve I_R ayrıldıktan sonra ortaya çıkan sol ve sağ düğümlerdir. Kesin teklif, tüm örnekleri her özelliğe göre sıralar ve olası tüm bölmeleri kullanır. Ancak, yinelemeli sıralama çok zaman alıcıdır. Sonuç olarak, yaygın olarak kullanılan bir strateji, özellik dağılımının yüzdelerine göre sınırlı bölünmüş adaylar önermektir. Aşırı uyumu önlemek için kullanılan iki yaygın teknik, büzülme ve özellik örneklemesidir. Büzülme, denklemdeki yaprak ağırlığını öğrenme oranı olarak adlandırılan hiper-parametresi η ile çarpar. Özellik örnekleme tekniği, pratikte etkili olduğu kanıtlanan her ağaç için bir özellik alt kümesini örneklemektedir (Chen ve Guestrin, 2016:785-794).

Gradyan Arttırılmış Ağaçlar yöntemini uygulanırken kullanılan model Şekil 18'deki gibidir. Ağırlık sayıları ve ağaç yapısının kümelemesinden dolayı şeklin boyutu küçük gözükmektedir.

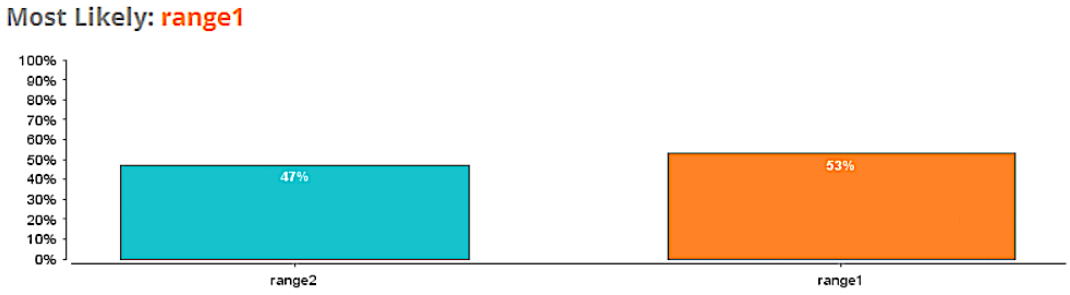
Şekil 18. Gradyan Arttırılmış Ağaçlar Modeli

Bu algoritmada kullanılan nitelikler ve ağırlıkları Çizelge 7'deki gibidir.

Çizelge 7 – Gradyan arttırılmış ağaçlar kullanılan nitelikler ve ağırlıkları

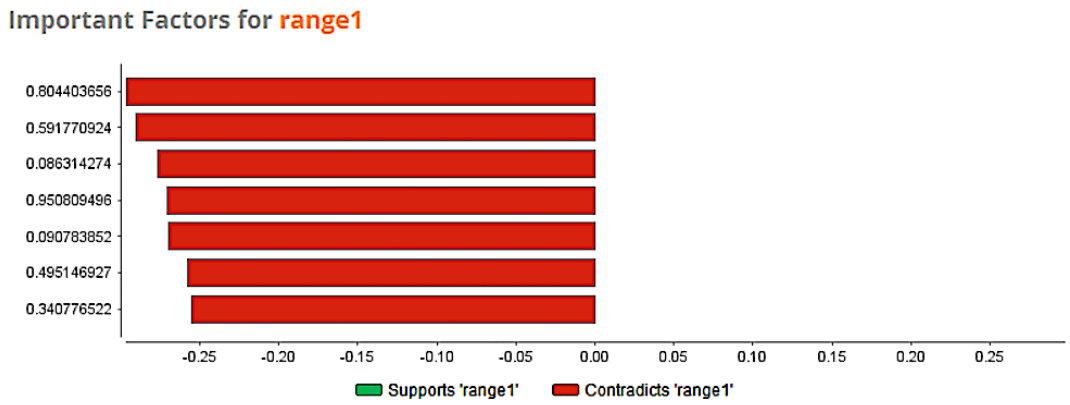
Nitelik	Ağırlık
0.329169068	0,2
0.04058144	0,1
0.304858309	0,1
0.340776522	0,1
0.350843754	0,1
0.950809496	0,1
0.090783852	0,1
0.218576167	0,1
0.591770924	0,1
0.215261493	0,1
0.804403656	0,1
0.09923022	0,1
0.086314274	0,1
0.937680009	0,1
0.788656772	0,1
0.495146927	0,1

Gradyan arttırılmış ağaçların simülasyonu sonucunda çıkabilecek muhtemel uzaklık Şekil 19'da gösterilmiştir.



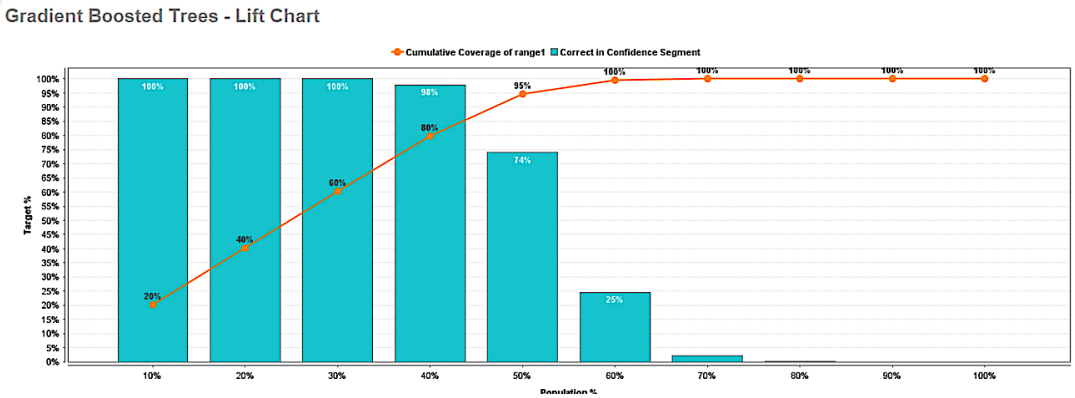
Şekil 19. Gradyan Arttırılmış Ağaçlar Simülasyon Gruplaması Sonucu

Bu muhtemel sonuç için önemli faktörler Şekil 20'deki gibidir.



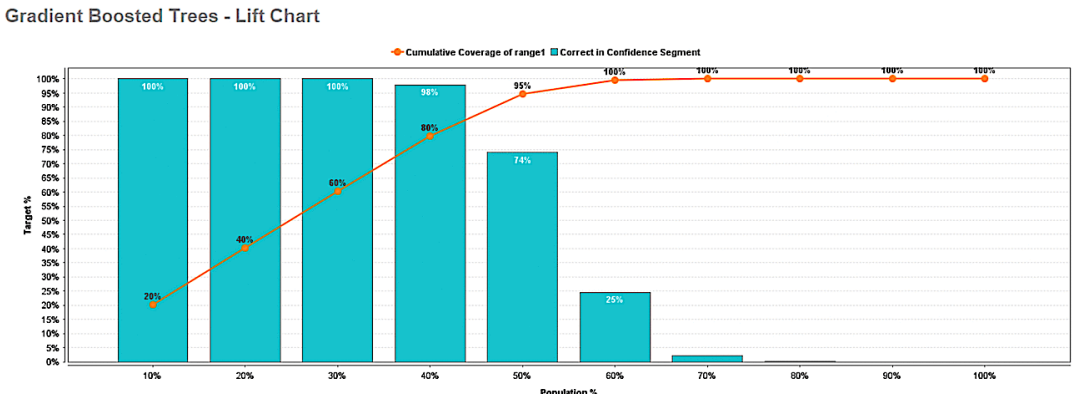
Şekil 20. Gradyan Arttırılmış Ağaçlar Uzaklık-1 için önemli faktörler

Bu algoritmanın yükseltme grafiği ise Şekil 21'deki gibidir.



Şekil 21. Gradyan Arttırılmış Ağaçlar için Yükseltme Grafiği

Gradyan arttırılmış ağaçların optimal parametreleri ise Şekil 22'de gösterilmiştir.



Şekil 22. Gradyan Arttırılmış Ağaçlar Optimal Parametreleri

Daha fazla özellik uygulamada genellikle daha yüksek tahmine dayalı doğruluk sağladığından, endüstriyel uygulamalarda kullanılan birçok veri kümesi genellikle çok fazla özellik içerir. GBT'nin, her yineleme sırasındaki tüm özellikler için gradyan

histogramlarının birleştirilmesini gerektirdiğini göz önünde bulundurarak, özelliklerin boyutu arttığında, model toplanmasının iletişim maliyeti de orantılı olarak artar. Bununla birlikte, mevcut sistemler verimsiz model toplamalarından dolayı bu yüksek boyutlu senaryoyu verimli bir şekilde ele almayı başaramamaktadır. Örneğin, MLlib (Url-17) eğitim sırasındaki parametreleri birleştirmek için harita azaltma çerçevesini kullanırken, XGBoost (Chen ve Guestrin, 2016:785-795) yerel çözümleri özetlemek için all-down MPI'yi seçmiştir. Oldukça büyük boyutta olan parametreler için, hem harita azaltma hem de tümü azaltma ile tek noktadan oluşan bir tıkanıklıkla karşılaşılır. Bu nedenle, model toplama paradigmaları yüksek boyutlu özellikler için uygun değildir. Harita küçültme ve azaltma işlemine bir alternatif, tek noktadan gelen tıkanmayı önlemek için parametreleri hizmet makineleriyle bölen parametre sunucusu mimarisidir. Ne yazık ki, Petuum (Wei J. , vd. 2015:381-394) ve TensorFlow (Abadi M., vd. 2016) gibi yaygın parametre sunucusu sistemleri, GBT'nin dağıtılmış eğitim süreci yukarıda bahsedilen ML algoritmalarından daha karmaşık olduğu için GBT için çözümler sunmamaktadır.

7. Destek Vektör Makinesi (Support Vector Machine – SVM)

Destek vektör makinesi, çok çeşitli alanlarda yüksek sınıflandırma doğruluğu nedeniyle birçok araştırmacının dikkatini çeken faydalı ve iyi bilinen denetimli öğrenme algoritmalarından biridir (Vapnik, 1995; Baesens ve ark. , 2003). SVM, doğrusal olmayan problemleri çözme yeteneğini kanıtlamıştır (Martens ve ark. 2009). Bununla birlikte, SVM, büyük veri uygulaması ile bellek ve zaman tüketiminin kullanımının artmasına neden olan yüksek işlemsel karmaşıklığa sahiptir. Doğrusal olmayan uygulamalarda çalışma prensibi, eğitim setini yüksek boyutlu özelliğe eşlemektir. Sınıflar arasında en uygun hiper düzlemi (hiper düzlemi ayırarak) inceleyerek maksimum bir sınır bulmak için doğrusal olarak ayrılabilir (Huang ve diğerleri, 2002).

Gauss radyal çekirdek fonksiyonu, genellikle doğrusal olmayan verileri yüksek boyutlu bir özellik uzayına aktarmak için kullanılır (Burbidge vd. 2001). Çekirdek işlevi,

$$k(\mathrm{x}_i.\mathrm{x}_j) = exp\left(-\frac{\left\|\mathrm{x}_i\text{-}\mathrm{x}_y\right\|^2}{2\sigma^2}\right) \quad \textbf{3-22}$$

Burada X_i, eğitim setinin girdisi olarak tanımlanır. Lemma kullanarak sınıflandırma araştırması için SVM ayırıcı hiperplanı gösterilebilir,

$$F(\mathrm{X}) = \mathrm{sgn}(\sum a_i y_i k(\mathrm{x}_i.\mathrm{x}_j) + b) \quad \textbf{3-23}$$

a_i, verileri yüksek boyutlu alana eşlemek için kullanılan Lagrang usulü olduğunda, y_i , $\mathbf{x}_i$ sınıfının etiketidir ve b bias, sgn ise örnekleri açıklayan işlevidir; yani, pozitif bir sınıf için + 1 veya negatif sınıf için -1'dir.

Destek vektör makinesi uygulanırken kullanılan model, aşağıdaki gibidir:

Çekirdek Model
Destek Vektörü Toplam Adedi: 2168
Tahminlerin şaşma değeri: -6.431

w[0.215261493] = 547973.246
w[0.788656772] = 530154.894
w[0.950809496] = 546957.686
w[0.591770924] = 539235.049
w[0.804403656] = 528858.216
w[0.350843754] = 526661.712
w[0.09923022] = 539813.643
w[0.04058144] = 532475.502
w[0.218576167] = 544046.364
w[0.090783852] = 536527.857
w[0.937680009] = 540722.249
w[0.495146927] = 533016.412
w[0.340776522] = 540139.120
w[0.304858309] = 542932.603

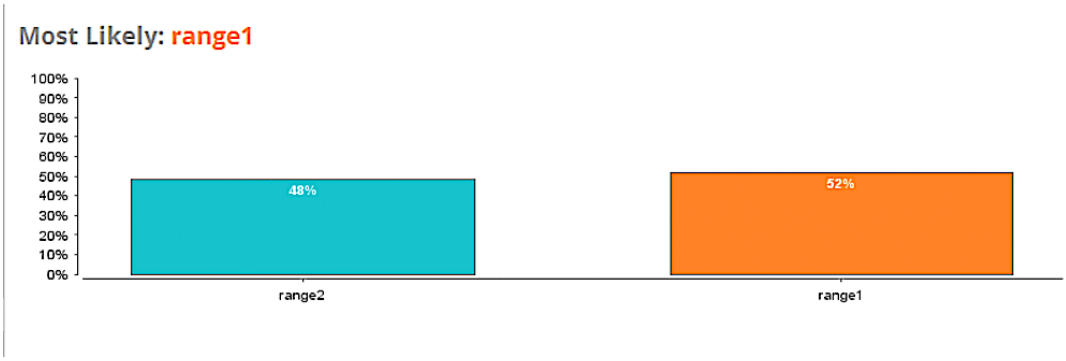
sınıf sayısı: 2
uzaklık1 için destek vektör sayısı: 1085
uzaklık2 için destek vektör sayısı: 1083

Destek vektör makinesinde kullanılan ağırlıklar Çizelge 8'de gösterilmiştir.

Çizelge 8 – Destek vektör makinesi nitelikleri ve ağırlıkları

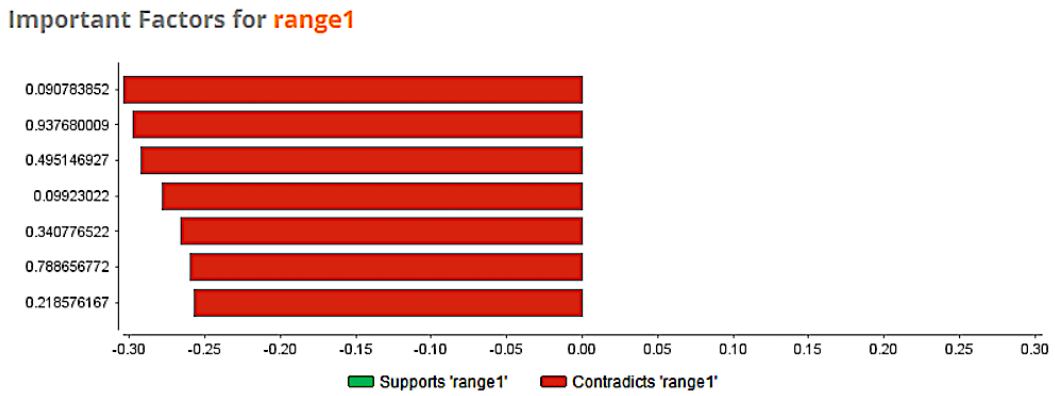
Nitelik	Ağırlık
0.090783852	0,1
0.218576167	0,1
0.350843754	0,1
0.304858309	0,1
0.340776522	0,1
0.215261493	0,1
0.788656772	0,1
0.04058144	0,1
0.804403656	0,1
0.495146927	0,1
0.937680009	0,1
0.950809496	0,1
0.591770924	0,1
0.09923022	0,1

Simülasyon gruplaması sonucu Şekil 23'de gösterilmiştir.



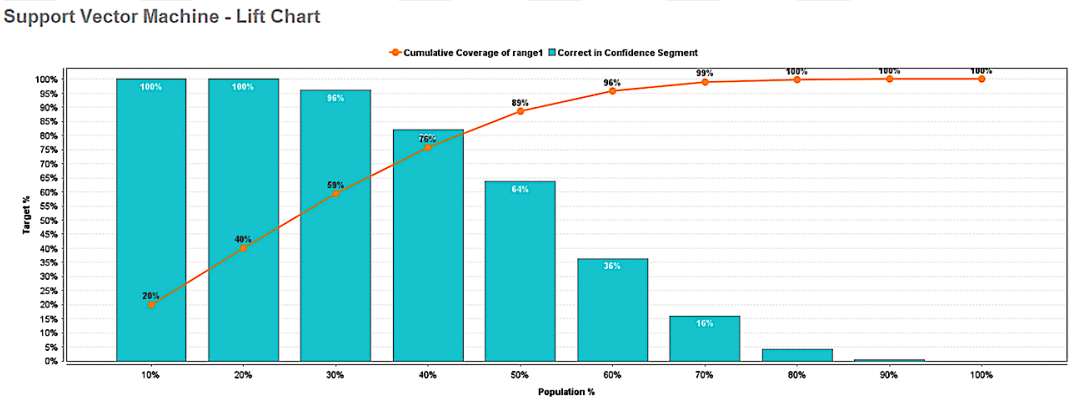
Şekil 23. Destek Vektör Makinesi Simülasyon Gruplaması Sonucu

Bu muhtemel sonuç için önemli faktörler Şekil 24'deki gibidir.



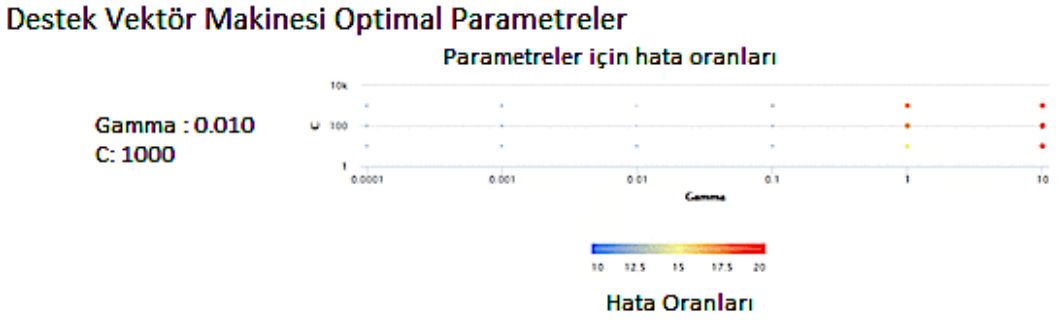
Şekil 24. Destek Vektör Makinesi Uzaklık-1 için önemli faktörler

Destek vektör makinesi yükseltme grafiği ise Şekil 25'deki gibidir.



Şekil 25. Destek Vektör Makinesi Yükseltme Grafiği

Destek vektör makinesinin optimal verileri Şekil 26'da gösterilmiştir.



Şekil 26. Destek Vektör Makinesi Optimal Parametreleri

8. Hızlı Büyük Marj (Fast Large Margin – FLM)

Hızlı Büyük Marj operatörü, R.E. Fan, K.W. Chang, C.J. Hsieh, X.R. Wang ve C.J. Lin tarafından önerilen lineer destek vektörü öğrenme düzenine dayanan bir hızlı

marj öğrenicisi uygular. Sonuç, klasik SVM veya lojistik regresyon uygulamaları tarafından sunulanlara benzer olmasına rağmen, bu doğrusal sınıflandırıcı, milyonlarca örnek ve nitelik içeren veri kümesi üzerinde çalışabilir. Standart SVM, bir giriş veri seti alır ve öngörülen her giriş için, iki olası sınıftan hangisinin girişi içerdiğini tahmin eder, bu da SVM'yi olasılıksız bir ikili doğrusal sınıflandırıcı yapar. Her biri iki kategoriden birine ait olarak işaretlenmiş bir dizi eğitim örneği göz önüne alındığında, bir SVM eğitim algoritması, bir kategoriye veya diğerine yeni örnekler atanan bir model oluşturur. Bir SVM modeli, örneklerin uzayda nokta olarak gösterilip, ayrı kategorilere ait örneklerin mümkün olduğunca geniş açık bir aralıkla bölünmesi için eşlenen bir temsilidir. Yeni örnekler daha sonra aynı alana eşleştirilir ve aralarındaki farkın hangi tarafında olduklarına dayanarak bir kategoriye ait olduğu tahmin edilir.

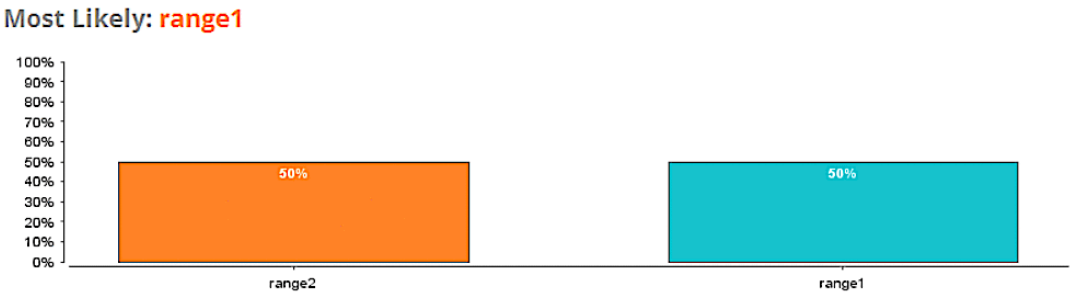
Bir destek vektörü makinesi, sınıflandırma, regresyon veya diğer görevler için kullanılabilen yüksek veya sonsuz boyutlu bir alanda bir hiper düzlem veya hiper düzlem seti oluşturur. Sezgisel olarak, herhangi bir sınıfın en yakın eğitim veri noktalarına (sözde işlevsel marj) en büyük mesafeye sahip olan hiper düzlem tarafından iyi bir ayrım elde edilir, çünkü genel olarak sınır marjı, sınıflandırıcının genelleme hatası azalır. Orijinal problem sonlu boyutlu bir uzayda belirtilebilse de, genellikle ayrımcılığa uğrayan kümelerin o uzayda doğrusal olarak ayrılmaması olur. Bu nedenle, orijinal sonlu boyutlu uzayın çok daha yüksek boyutlu bir alana haritalanması ve muhtemelen bu alanda ayrılmayı kolaylaştırması önerildi. Hesaplama yükünü makul tutmak için, SVM şemaları tarafından kullanılan haritalama, seçilen bir çekirdek fonksiyonu $K\left(x, y\right)$ cinsinden tanımlanarak, nokta ürünlerin orijinal alandaki değişkenler açısından kolayca hesaplanmasını sağlamak üzere tasarlanmıştır. Yüksek boyutlu uzaydaki hiper düzlemler, iç çarpımı o uzaydaki bir vektör ile sabit olan noktaların kümesi olarak tanımlanır. Hızlı büyük marj operatörü bu işlemleri çok büyük veri setlerinde uygulayabildiği için tercih sebebi olmuştur.

Hızlı Büyük Marj'da kullanılan ağırlıklar Çizelge 9'de verilmiştir.

Çizelge 9 – Hızlı büyük marj algoritması nitelikleri ve ağırlıkları

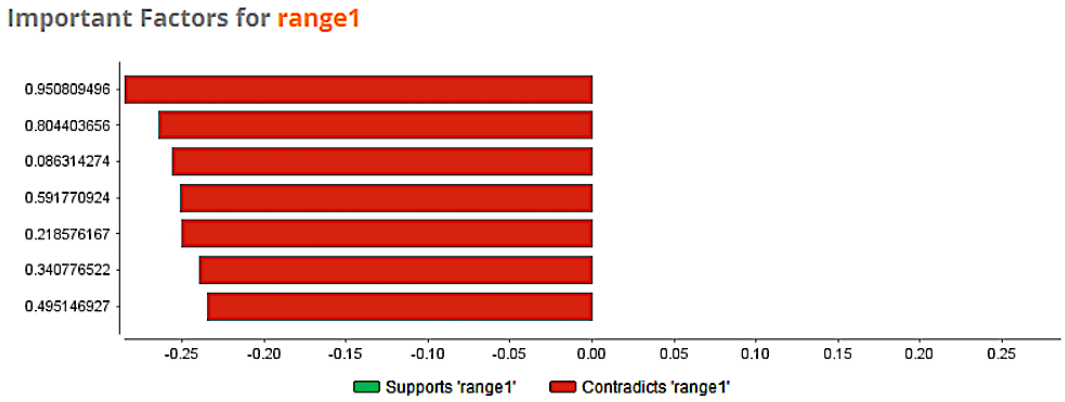
Nitelik	Ağırlık
0.329169068	0,2
0.04058144	0,1
0.304858309	0,1
0.340776522	0,1
0.350843754	0,1
0.950809496	0,1
0.090783852	0,1
0.218576167	0,1
0.591770924	0,1
0.09923022	0,1
0.215261493	0,1
0.804403656	0,1
0.086314274	0,1
0.937680009	0,1
0.788656772	0,1
0.495146927	0

Simülasyon gruplaması sonucu Şekil 27'deki gibidir.



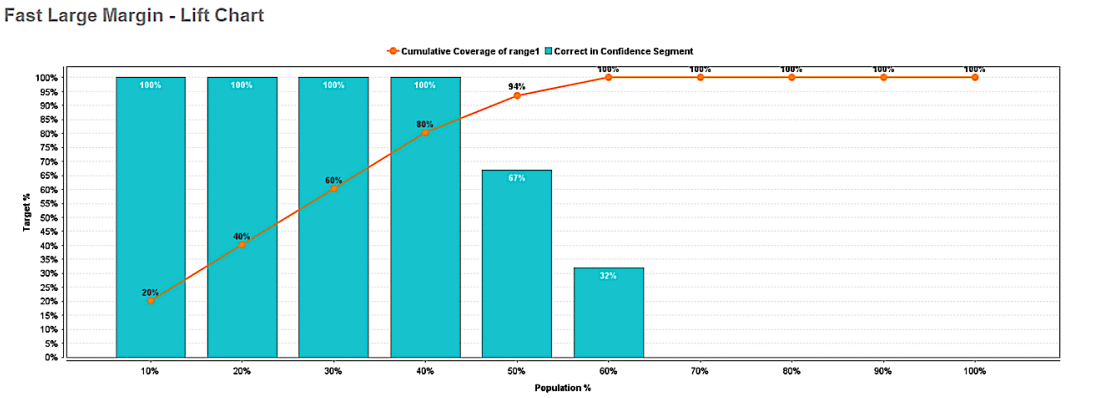
Şekil 27. Hızlı Büyük Marj algoritması Simülasyon gruplaması sonucu

Bu muhtemel sonuç için önemli faktörler Şekil 28'deki gibidir.



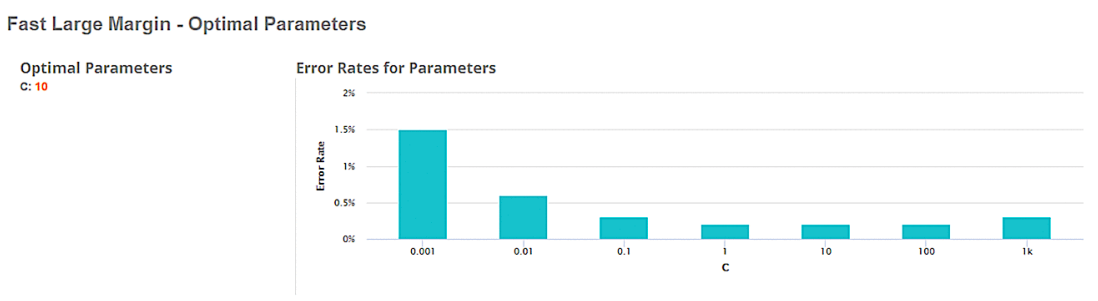
Şekil 28. Hızlı Büyük Marj Algoritması Uzaklık-1 için önemli faktörler

Hızlı büyük marj'a ait yükseltme grafiği Şekil 29'da gösterilmiştir.



Şekil 29. Hızlı Büyük Marj Algoritması Yükseltme Grafiği

Bu algoritmanın optimal parametreleri ise Şekil 30'da gösterilmiştir.



Şekil 30. Hızlı Büyük Marj Algoritması Optimal Parametreleri

9. Konvolüsyonel (Evrişimli) Sinir Ağı (Convolutional Neural Network – CNN)

Derin Sinir Ağı, çok sayıda gizli katmana sahip Yapay Sinir Ağıdır (YSA) ve çok miktarda veri işleme yeteneğine sahiptir. Konvolüsyonel Sinir Ağı, en eski ve en

popüler derin sinir ağlarından biridir. Adı, evrişim adı verilen matrisler arasındaki doğrusal operatörden gelir. Bir CNN çoklu katmanlardan oluşur: evrişim katmanı, havuz katmanı, tamamen bağlı katman ve doğrusal olmayan katman. CNN'ler özellikle görüntü sınıflandırma, bilgisayarla görme ve doğal dil işleme alanlarında umut verici bir şekilde performans gösteriyor. Bu çalışmada, CNN'lerin bilgisayar vizyonunda popülerliğini artıran ilk mimari olan AlexNet kullanılmıştır. AlexNet'te, değişen evrişimsel tabakalar ve havuzlama tabakaları yerine tüm evrişimsel tabakalar bir araya getirilmiştir. Bu nedenle, LeNet ve onun geliştirilmiş versiyonu ile karşılaştırıldığında daha fazla derinlik ve boyut sağlar.

Kullanılan model şu şekildedir:

Model Metrikleri Türü: Binom

Açıklama: 10040 örnek ile geçici eğitim çerçevesinde rapor edilen metrikler

model numarası: rm-h2o-model-model-390517

çerçeve numarası: rm-h2o-frame-model-615009.temporary.sample.28.49%

MSE: 0.0026381898

R^2: 0.98944676

AUC: 0.99996436

logloss: 0.009106332

KM: Karmaşıklık Matrisi (dikey: gerçek; karşısında: öngörülen):

Uzaklık2	Uzaklık1	Hata	Oran
Uzaklık2	4964	22	0.0044 = 22 / 4,986
Uzaklık1	6	5048	0.0012 = 6 / 5,054
Toplam	4970	5070	0.0028 = 28 / 10,040

Kazanç/Kaldırma Çizelgesu (Avg response rate: 50.34 %):

Nöron Katmanlarının durumu (tahmin 0, 2-sınıf sınıflandırma, bernoulli dağıtımı, CrossEntropy kaybı, 3,502 Ağırlıklar/bias, 47.5 KB, 351,000 eğitim örneği, mini-parti boyutu 1):

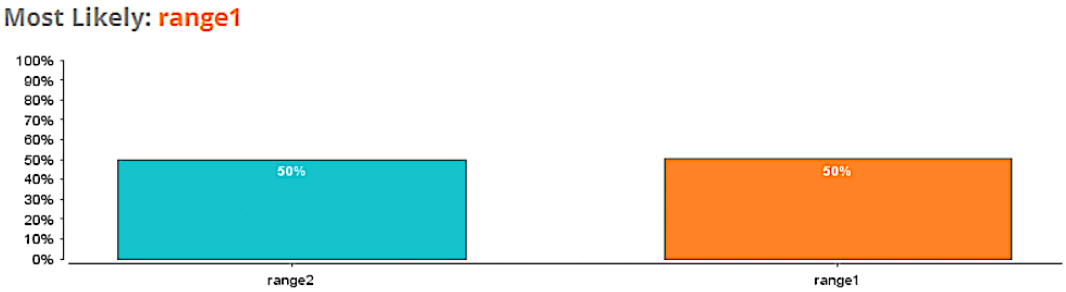
H2O versiyon: 3.8.2.6-rm9.0.0

Kullanılan ağırlıklar, Çizelge 10'da verilmiştir.

Çizelge 10 – Konvolüsyonel sinir ağında kullanılan nitelikler ve ağırlıkları

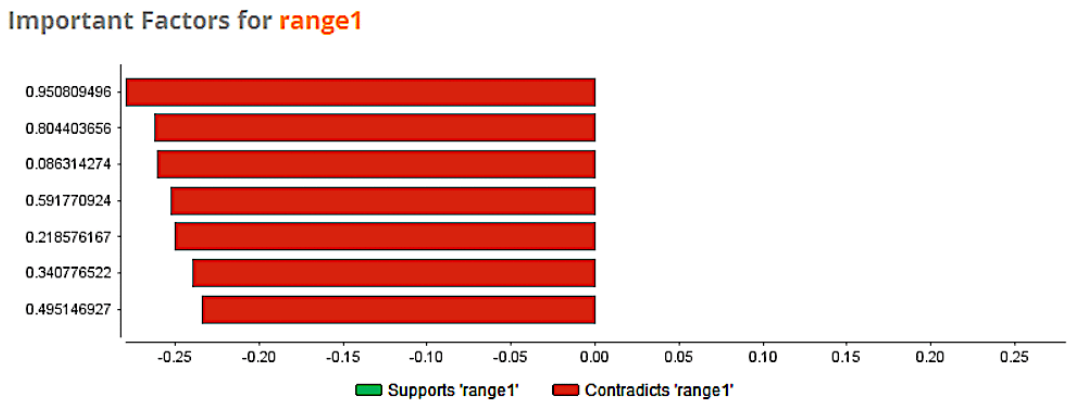
Nitelik	Ağırlık
0.329169068	0,2
0.04058144	0,1
0.304858309	0,1
0.340776522	0,1
0.350843754	0,1
0.950809496	0,1
0.090783852	0,1
0.218576167	0,1
0.591770924	0,1
0.09923022	0,1
0.215261493	0,1
0.804403656	0,1
0.086314274	0,1
0.937680009	0,1
0.788656772	0,1
0.495146927	0

Simülasyon gruplaması sonrası çıkan değer Şekil 31'de gösterilmiştir.



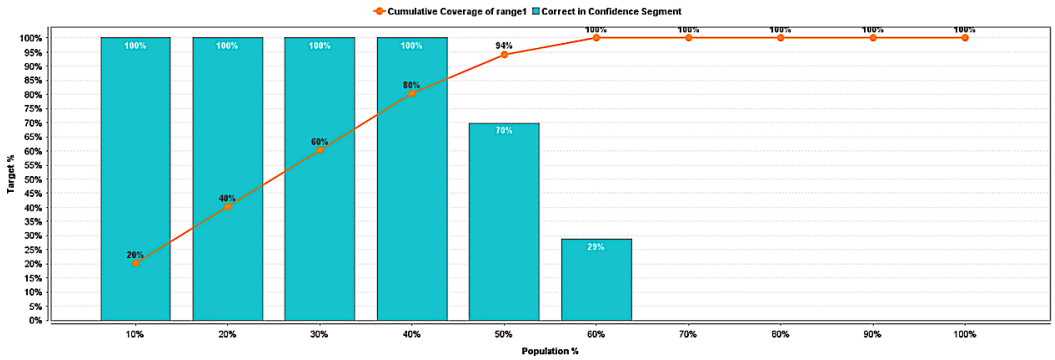
Şekil 31. Konvolüsyonel Sinir Ağı Simülasyon Gruplaması Sonucu

Uzaklık-1 için önemli faktörler Şekil 32'de gösterilmiştir.



Şekil 32. Konvolüsyonel Sinir Ağı Uzaklık-1 için önemli faktörler

Bu algoritma için yükseltme grafiği Şekil 33'deki gibidir.



Şekil 33. Konvolüsyonel Sinir Ağı Yükseltme Grafiği

Sınıflandırmada sinir ağlarının iyi olmasının matematiksel nedeni, evrensel yaklaşım teoremidir. Bir sinir ağının, kompakt bir alt kümede sürekli gerçek değerli herhangi bir işlevi yerine getirebileceğini belirtir. Yaklaşımın kalitesi nöronların sayısına bağlıdır. Ayrıca nöronları mevcut katmanlara eklemek yerine ek katmanlara eklemenin yaklaştırma kalitesini daha hızlı arttırdığı gösterilmiştir. Ayrıca sinir ağları, verilerdeki karmaşık desenleri bile açıklayabilen işlevleri oluşturma yeteneğine sahip olduğu için de tahmin uygulamalarında kullanılmaktadır.

IV. BULGULAR

A. Naif Bayes (Naive Bayes – NB)

Naif Bayes modeli performansları ve tahmin sonuçları aşağıdaki şekillerde gösterilmiştir. Değerlendirilen özellik kümesi sayısı 425 ve üretilen özellik sayısı 37'dir. Naif Bayes performans sonuçlarından ilki Şekil 34'te gösterilmiştir.

Maliyet ve Faydalar	
Toplam Maliyet/Fayda:	25432
Ortalama Maliyet / Fayda:	0.978
En iyi seçenek için toplam(uzaklık2) :	72
Kazanç:	25360

Şekil 34. Naif Bayes Maliyet ve Faydaları

Naif Bayes Algoritması Hata Matrisinden gelen sonuçlar Çizelge 11'deki gibidir

Çizelge 11 - Naif bayes hata matrisi sonuçları

Kriter	Değer	Standart Sapma
Doğruluk	98.9%	±0.1%
Sınıflandırma Hatası	1.1%	±0.1%
Eğrinin Altındaki Alan	100.0%	±0.0%
Hassasiyet	98.1%	±0.1%
Geri Çağırma	99.8%	±0.1%
Testin Doğruluğu	98.9%	±0.1%
Gerçek Pozitifler Oranı	99.8%	±0.1%
Belirlilik	98.1%	±0.1%

Naif Bayes Algoritmasının Hata Matrisi Değerleri Çizelge 12'deki gibidir.

Çizelge 12. Naif bayes hata matrisi değerleri

Kriter	Değer	Standart Sapma
Doğruluk	98.9%	±0.1%
Sınıflandırma Hatası	1.1%	±0.1%
Eğrinin Altındaki Alan	100.0%	±0.0%
Hassasiyet	98.1%	±0.1%
Geri Çağırma	99.8%	±0.1%
Testin Doğruluğu	98.9%	±0.1%
Gerçek Pozitifler Oranı	99.8%	±0.1%
Belirlilik	98.1%	±0.1%

B. Genelleştirilmiş Doğrusal Model (Generalized Linear Model – GLM)

Genelleştirilmiş Doğrusal Model performansları ve tahmin sonuçları aşağıda gösterilmiştir. Değerlendirilen özellik kümesi sayısı 722 ve üretilen özellik sayısı 55'tir. Genelleştirilmiş Doğrusal Model performans sonuçlarından ilki Şekil 35'te gösterilmiştir.

Maliyet ve Faydalar	
Toplam Maliyet/Fayda:	25698
Ortalama Maliyet / Fayda:	0.988
En iyi seçenek için toplam(uzaklık2) :	72
Kazanç:	25626

Şekil 35. Genelleştirilmiş Doğrusal Model Maliyet ve Faydaları

Genelleştirilmiş Doğrusal Model Algoritması Hata Matrisinden gelen sonuçlar Çizelge 13'teki gibidir.

Çizelge 13 - Genelleştirilmiş doğrusal model hata matrisi sonuçları

Kriter	Değer	Standart Sapma
Doğruluk	98.9%	±0.1%
Sınıflandırma Hatası	1.1%	±0.1%
Eğrinin Altındaki Alan	100.0%	±0.0%
Hassasiyet	98.1%	±0.1%
Geri Çağırma	99.8%	±0.1%
Testin Doğruluğu	98.9%	±0.1%
Gerçek Pozitifler Oranı	99.8%	±0.1%
Belirlilik	98.1%	±0.1%

Genelleştirilmiş Doğrusal Model Algoritmasının Hata Matrisi Değerleri Çizelge 14'teki gibidir.

Çizelge 14 - Genelleştirilmiş doğrusal model hata matrisi değerleri

Hata Matrisi	Asıl ağırlık2	Asıl ağırlık1	Sınıf hassasiyeti
Tahmini uzaklık2	9183	0	100.00%
Tahmini uzaklık1	110	9279	98.83%
Sınıf geri çağırma	98.82%	100.00%	

C. Lojistik Regresyon (Logistic Regression – LR)

Lojistik Regresyon performansları ve tahmin sonuçları aşağıda gösterilmiştir. Değerlendirilen özellik kümesi sayısı 1164 ve üretilen özellik sayısı 79'dur. Lojistik Regresyon performans sonuçlarından ilki Şekil 36'da gösterilmiştir.

Maliyet ve Faydalar	
Toplam Maliyet/Fayda:	21,930
Ortalama Maliyet / Fayda:	0.843
En iyi seçenek için toplam(uzaklık2) :	72
Kazanç:	21,858

Şekil 36. Lojistik Regresyon Maliyet ve Faydaları

Lojistik Regresyon Algoritması Hata Matrisinden gelen sonuçlar Çizelge 15'teki gibidir.

Çizelge 15 - Lojistik regresyon hata matrisi sonuçları

Kriter	Değer	Standart Sapma
Doğruluk	92.1%	±0.3%
Sınıflandırma Hatası	7.9%	±0.3%
Eğrinin Altındaki Alan	99.6%	±0.0%
Hassasiyet	86.4%	±0.6%
Geri Çağırma	100.0%	±0.0%
Testin Doğruluğu	92.7%	±0.4%
Gerçek Pozitifler Oranı	100.0%	±0.0%
Belirlilik	84.4%	±0.6%

Lojistik Regresyon Algoritmasının Hata Matrisi Değerleri Çizelge 16'daki gibidir.

Çizelge 16. Lojistik regresyon hata matrisi değerleri

Kriter	Değer	Standart Sapma
Doğruluk	92.1%	±0.3%
Sınıflandırma Hatası	7.9%	±0.3%
Eğrinin Altındaki Alan	99.6%	±0.0%
Hassasiyet	86.4%	±0.6%
Geri Çağırma	100.0%	±0.0%
Testin Doğruluğu	92.7%	±0.4%
Gerçek Pozitifler Oranı	100.0%	±0.0%
Belirlilik	84.4%	±0.6%

D. Karar Ağaçları (Decision Trees – DT)

Karar Ağaçları performansları ve tahmin sonuçları aşağıda gösterilmiştir. Değerlendirilen özellik kümesi sayısı 1223 ve üretilen özellik sayısı 95'tir. Karar Ağaçları performans sonuçlarından ilki Şekil 37'de gösterilmiştir.

Maliyet ve Faydalar	
Toplam Maliyet/Fayda :	-1646
Ortalama Maliyet / Fayda:	-0.063
En iyi seçenek için toplam(uzaklık2) :	72
Kazanç:	-1,718

Şekil 37. Karar Ağacı Maliyet ve Faydaları

Karar Ağaçları Algoritması Hata Matrisinden gelen sonuçlar Çizelge 17'deki gibidir.

Çizelge 17 - Karar ağacı hata matrisi sonuçları

Kriter	Değer	Standart Sapma
Doğruluk	46.9%	±0.2%
Sınıflandırma Hatası	53.1%	±0.2%
Eğrinin Altındaki Alan	46.8%	±0.4%
Hassasiyet	34.3%	±2.0%
Geri Çağırma	7.0%	±0.4%
Testin Doğruluğu	11.6%	±0.6%
Gerçek Pozitifler Oranı	7.0%	±0.4%
Belirlilik	86.7%	±0.5%

Karar Ağaçları Algoritmasının Hata Matrisi Değerleri Çizelge 18'deki gibidir.

Çizelge 18 - Karar ağacı hata matrisi değerleri

Hata Matrisi	Asıl ağırlık2	Asıl ağırlık1	Sınıf hassasiyeti
Tahmin uzaklık2	8061	8623	48.32%
Tahmin uzaklık1	1240	647	34.29%
Sınıf geri çağırma	86.67%	6.98%	

E. Rastgele Orman (Random Forest – RF)

Rastgele Orman performansları ve tahmin sonuçları aşağıda gösterilmiştir. Değerlendirilen özellik kümesi sayısı 911 ve üretilen özellik sayısı 82'dir. Random Forest performans sonuçlarından ilki Şekil 38'de gösterilmiştir.

Maliyet ve Faydalar	
Toplam Maliyet/Fayda:	2652
Ortalama Maliyet / Fayda:	0.106
En iyi seçenek için toplam(uzaklık2) :	70
Kazanç:	2582

Şekil 38. Rastgele Orman Maliyet ve Faydaları

Rastgele Orman Algoritması Hata Matrisinden gelen sonuçlar Çizelge 19'daki gibidir

Çizelge 19 - Rastgele orman hata matrisi sonuçları

Kriter	Değer	Standart Sapma
Doğruluk	55.4%	±0.3%
Sınıflandırma Hatası	44.6%	±0.3%
Eğrinin Altındaki Alan	68.8%	±1.0%
Hassasiyet	53.0%	±0.6%
Geri Çağırma	94.5%	±0.4%
Testin Doğruluğu	67.9%	±0.4%
Gerçek Pozitifler Oranı	94.5%	±0.4%
Belirlilik	16.4%	±0.9%

Rastgele Orman Algoritmasının Hata Matrisi Değerleri Çizelge 20'deki gibidir.

Çizelge 20 - Rastgele orman hata matrisi değerleri

Hata Matrisi	Asıl ağırlık2	Asıl ağırlık1	Sınıf hassasiyeti
Tahmin uzaklık2	1467	489	75.00%
Tahmin uzaklık1	7472	8429	53.01%

Sınıf geri çağırma	16.41%	94.52%

F. Gradyan Arttırılmış Ağaçlar (Gradient Boosted Trees – GBT)

Gradyan Arttırılmış Ağaçlar performansları ve tahmin sonuçları aşağıda gösterilmiştir. Değerlendirilen özellik kümesi sayısı 296 ve üretilen özellik sayısı 26'dır. Gradyan Arttırılmış Ağaçlar performans sonuçlarından ilki Şekil 39'da gösterilmiştir.

Maliyet ve Faydalar	
Toplam Maliyet/Fayda:	21964
Ortalama Maliyet / Fayda:	0.879
En iyi seçenek için toplam(uzaklık2) :	70
Kazanç:	21894

Şekil 39. Gradyan Arttırılmış Ağaçlar Maliyet ve Faydaları

Gradyan Arttırılmış Ağaçlar Algoritması Hata Matrisinden gelen sonuçlar Çizelge 21'deki gibidir.

Çizelge 21 - Gradyan arttırılmış ağaçlar hata matrisi sonuçları

Kriter	**Değer**	**Standart Sapma**
Doğruluk	94.0%	±0.4%
Sınıflandırma Hatası	6.0%	±0.4%
Eğrinin Altındaki Alan	99.1%	±0.1%
Hassasiyet	91.7%	±0.8%
Geri Çağırma	96.7%	±0.4%
Testin Doğruluğu	94.1%	±0.5%
Gerçek Pozitifler Oranı	96.7%	±0.4%
Belirlilik	91.2%	±0.7%

Gradyan Arttırılmış Ağaçlar Algoritmasının Hata Matrisi Değerleri Çizelge 22'deki gibidir.

Çizelge 22 - Gradyan arttırılmış ağaçlar hata matrisi değerleri

Hata Matrisi	**Asıl ağırlık2**	**Asıl ağırlık1**	**Sınıf hassasiyeti**
Tahmin uzaklık2	8154	295	96.51%
Tahmin uzaklık1	785	8623	91.66%
Sınıf geri çağırma	91.22%	96.69%	

G. Destek Vektör Makinesi (Support Vector Machine – SVM)

Destek Vektör Makinesi performansları ve tahmin sonuçları aşağıda gösterilmiştir. Değerlendirilen özellik kümesi sayısı 2907 ve üretilen özellik sayısı 177'dir. Destek Vektör Makinesi performans sonuçlarından ilki Şekil 40'ta gösterilmiştir.

Maliyet ve Faydalar	
Toplam Maliyet/Fayda:	4600
Ortalama Maliyet / Fayda:	0.767
En iyi seçenek için toplam(uzaklık2) :	16
Kazanç:	4584

Şekil 40. Destek Vektör Makinesi Maliyet ve Faydaları

Destek Vektör Makinesi Algoritması Hata Matrisinden gelen sonuçlar Çizelge 23'teki gibidir.

Çizelge 23 - Destek vektör makinesi hata matrisi sonuçları

Kriter	Değer	Standart Sapma
Doğruluk	88.4%	±0.7%
Sınıflandırma Hatası	11.6%	±0.7%
Eğrinin Altındaki Alan	95.9%	±0.5%
Hassasiyet	87.1%	±0.6%
Geri Çağırma	90.2%	±1.3%
Testin Doğruluğu	88.6%	±0.7%
Gerçek Pozitifler Oranı	90.2%	±1.3%
Belirlilik	86.6%	±0.9%

Destek Vektör Makinesi Algoritmasının Hata Matrisi Değerleri Çizelge 24'teki gibidir.

Çizelge 24 - Destek vektör makinesi hata matrisi değerleri

Hata Matrisi	Asıl ağırlık2	Asıl ağırlık1	Sınıf hassasiyeti
Tahmin uzaklık2	1857	210	89.84%
Tahmin uzaklık1	287	1932	87.07%
Sınıf geri çağırma	86.61%	90.20%	

H. Hızlı Büyük Marj (Fast Large Margin – FLM)

Hızlı Büyük Marj performansları ve tahmin sonuçları aşağıda gösterilmiştir. Değerlendirilen özellik kümesi sayısı 375 ve üretilen özellik sayısı 32'dir. Hızlı Büyük Marj performans sonuçlarından ilki Şekil 41'de gösterilmiştir.

Maliyet ve Faydalar	
Toplam Maliyet/Fayda:	24100
Ortalama Maliyet / Fayda:	0.927
En iyi seçenek için toplam(uzaklık2) :	72
Kazanç:	24028

Şekil 41. Hızlı Büyük Marj Maliyet ve Faydaları

Hızlı Büyük Marj Algoritması Hata Matrisinden gelen sonuçlar Çizelge 25'teki gibidir.

Çizelge 25 - Hızlı büyük marj hata matrisi sonuçları

Kriter	Değer	Standart Sapma
Doğruluk	96.3%	±0.1%
Sınıflandırma Hatası	3.7%	±0.1%
Eğrinin Altındaki Alan	99.8%	±0.0%
Hassasiyet	93.0%	±0.3%
Geri Çağırma	100.0%	±0.0%
Testin Doğruluğu	96.4%	±0.1%
Gerçek Pozitifler Oranı	100.0%	±0.0%
Belirlilik	92.6%	±0.1%

Hızlı Büyük Marj Algoritmasının Hata Matrisi Değerleri Çizelge 26'daki gibidir.

Çizelge 26 - Hızlı büyük marj hata matrisi değerleri

Hata Matrisi	Asıl ağırlık2	Asıl ağırlık1	Sınıf hassasiyeti
Tahmin uzaklık2	8630	0	100.00%
Tahmin uzaklık1	694	9248	93.02%
Sınıf geri çağırma	92.56%	100.00%	

İ. Konvolüsyonel Sinir Ağları (Convolutional Neural Networks – CNN)

Konvolüsyonel Sinir Ağı performansları ve tahmin sonuçları aşağıda gösterilmiştir. Değerlendirilen özellik kümesi sayısı 339 ve üretilen özellik sayısı 32'dir. Konvolüsyonel Sinir Ağları performans sonuçlarından ilki Şekil 42'de gösterilmiştir.

Maliyet ve Faydalar	
Toplam Maliyet/Fayda:	22274
Ortalama Maliyet / Fayda:	0.857
En iyi seçenek için toplam(uzaklık2) :	72
Kazanç:	22202

Şekil 42. Konvolüsyonel Sinir Ağları Maliyet ve Faydaları

Konvolüsyonel Sinir Ağları Algoritması Hata Matrisinden gelen sonuçlar Çizelge 27'deki gibidir.

Çizelge 27 - Konvolüsyonel sinir ağları hata matrisi sonuçları

Kriter	Değer	Standart Sapma
Doğruluk	92.8%	±0.3%
Sınıflandırma Hatası	7.2%	±0.3%
Eğrinin Altındaki Alan	100.0%	±0.0%
Hassasiyet	87.4%	±0.5%
Geri Çağırma	100.0%	±0.0%
Testin Doğruluğu	93.3%	±0.3%
Gerçek Pozitifler Oranı	100.0%	±0.0%
Belirlilik	85.7%	±0.5%

Konvolüsyonel Sinir Ağları Algoritmasının Hata Matrisi Değerleri Çizelge 28'deki gibidir.

Çizelge 28 - Konvolüsyonel sinir ağları hata matrisi değerleri

Hata Matrisi	Asıl ağırlık2	Asıl ağırlık1	Sınıf hassasiyeti
Tahmin uzaklık2	7991	0	100.00%
Tahmin uzaklık1	1333	9248	87.40%
Sınıf geri çağırma	85.70%	100.00%	

V. SONUÇ VE ÖNERİLER

Bu bölümde, kullanılan algoritmalara ait sonuçlar ve karşılaştırmalar gösterilmiştir. Toplamda 33,321 adet model üretilmiştir. Algoritmalara ait sonuçlar, hata matrisinin formüllerine göre hesaplanmış, nasıl hesaplandıkları gösterilmiş ve veriler paylaşılmış ardından grafiklendirilmiştir. TP, TN, FP, FN sırasıyla gerçek pozitif, gerçek negatif, yanlış pozitif ve yanlış negatif anlamına gelir. True Positive (TP), False Positive (FP), True Negative (TN) ve False Negative'in (FN) neye karşılık geldiği aşağıda açıklanmıştır:

Gerçek pozitif (TP): Sunucu yükünün bilindiği ve modelin de bildiği şekilde öngördüğü durumdur.

Gerçek negatif (TN): Sunucu yükünün bilinmediği ve modelin de bilinmediği şekilde öngördüğü durumdur.

Yanlış pozitif (FP): Sunucu yükünün bilinmediği ama modelin bildiği şekilde öngördüğü durumdur.

Yanlış negatif (FN): Sunucu yükünün bilindiği ama modelin bilinmediği şekilde öngördüğü durumdur.

$$Accuracy(Doğruluk) = \frac{(\mathrm{TP} + \mathrm{TN})}{(\mathrm{TP} + \mathrm{FP} + \mathrm{FN} + \mathrm{TN})} \qquad 5-1$$

$$Precision(Hassasiyet) = \frac{(\mathrm{TP})}{(\mathrm{TP} + \mathrm{FP})} \qquad 5-2$$

$$Recall(Geri\ Çağırma) = \frac{(\mathrm{TP})}{(\mathrm{TP} + \mathrm{FN})} \qquad 5-3$$

$$Sensitivity(Gerçek\ Pozitifler\ Oranı) = \frac{(\mathrm{TP})}{(\mathrm{TP} + \mathrm{FN})} \qquad 5-4$$

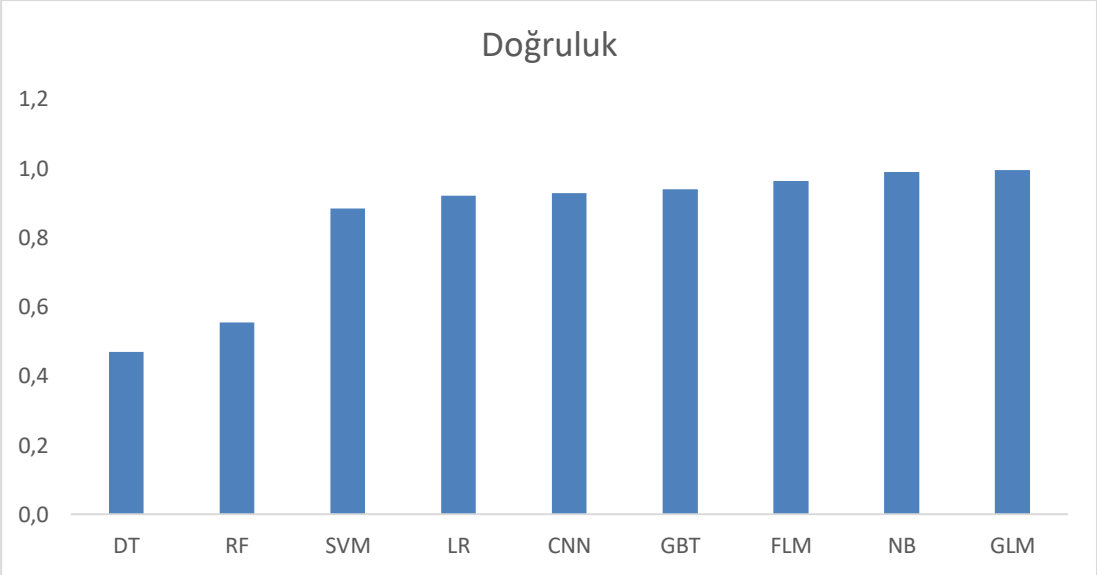
$$Classification\ Error(Sınıflandırma\ Hatası) = \frac{(\mathrm{FP} + \mathrm{FN})}{(\mathrm{TP} + \mathrm{FP} + \mathrm{FN} + \mathrm{TN})} \qquad 5-5$$

$$Specificity(Belirlilik) = \frac{(\mathrm{TN})}{(\mathrm{TN} + \mathrm{FP})} \qquad 5-6$$

$$F\ Measure(Deneyin\ Doğruluğu) = \frac{(2 * \mathrm{TP})}{(2 * \mathrm{TP} + \mathrm{FP} + \mathrm{FN})} \qquad 5-7$$

Area Under Curve (Eğri Altındaki Alan), ikili sınıflandırma problemi için kullanılır. Sınıflandırıcının AUC'si, sınıflandırıcının rastgele seçilen bir negatif örnekten daha yüksek olarak rastgele seçilen bir pozitif örnek sıralama olasılığına eşittir. İki nokta arasındaki bir eğri altındaki alan, iki nokta arasında belirli bir integral yaparak bulunabilir. Y = f (x) eğrisinin altındaki alanı x = a ve x = b arasında bulmak için, integralin a ve b sınırları arasında y = f (x) şeklinde yapılması gerekir. Grafikler oluşturulurken, düşükten yükseğe göre sıralanmışlardır.

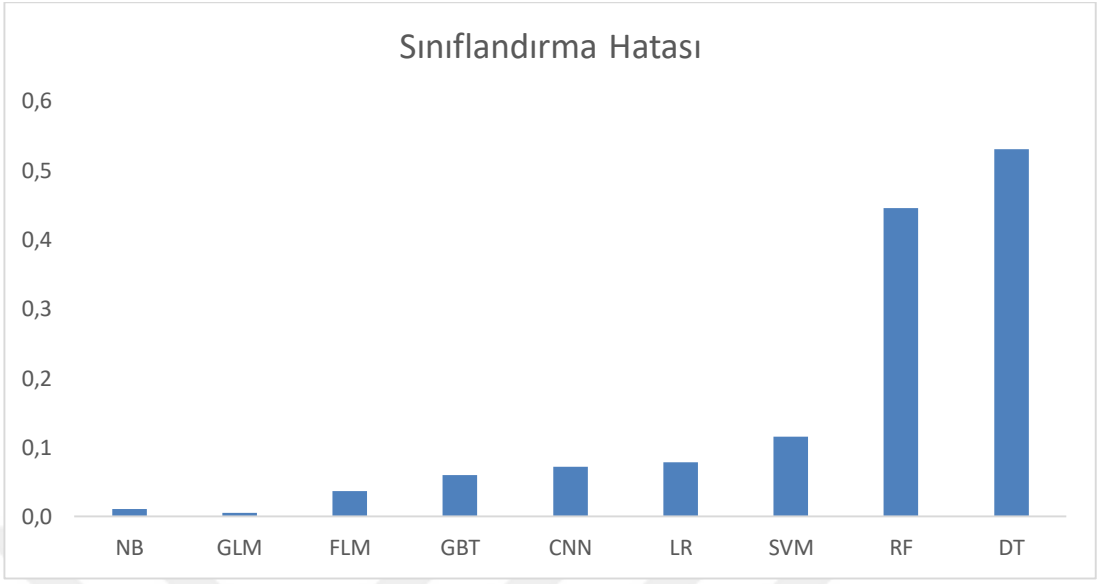
A. Doğruluk (Accuracy) Analizi



Şekil 43. Doğruluğa Dayalı Performans Analizi

Doğruluğa Dayalı Performans Analizinde Genelleştirilmiş Doğrusal Model en iyi performansı göstermiştir ve ardından Naif Bayes, Hızlı Büyük Marj gelmiştir.

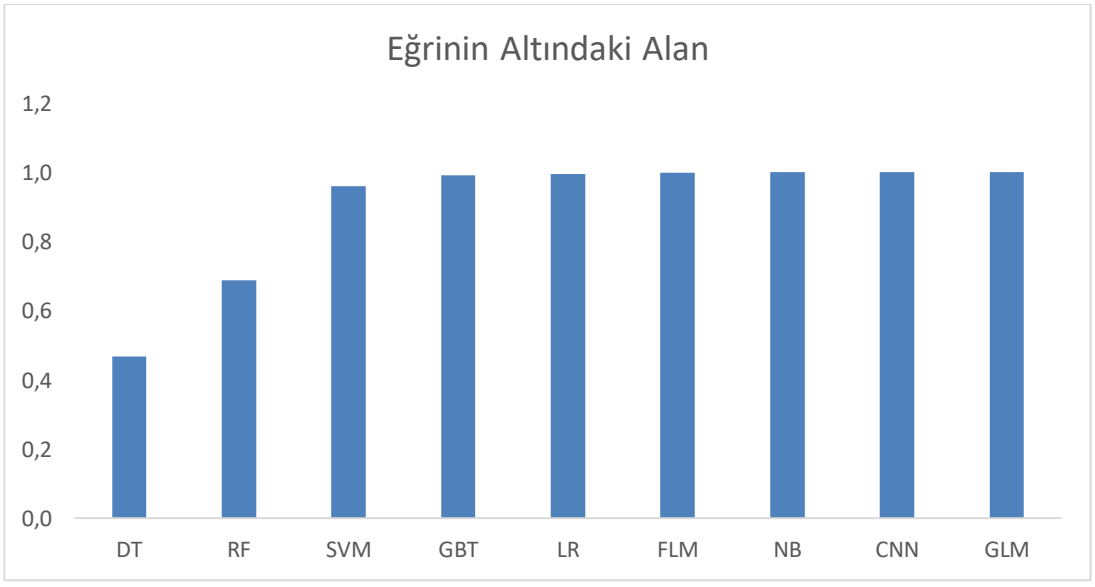
B. Sınıflandırma Hatası (Classification Error) Analizi



Şekil 44. Sınıflandırma Hatasına Dayalı Performans Analizi

Sınıflandırma Hatasına Dayalı Performans Analizinde Genelleştirilmiş Doğrusal Model en iyi performansı göstermiştir yani en az hata oranına sahiptir ve ardından Naif Bayes, Hızlı Büyük Marj gelmiştir.

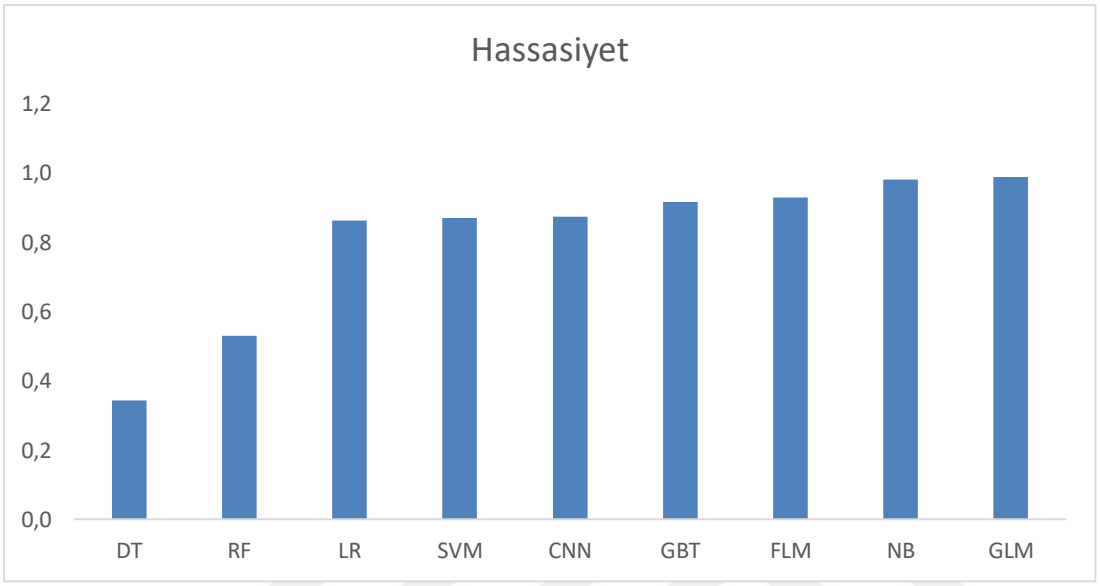
C. Eğrinin Altındaki Alan (AUC) Analizi



Şekil 45. Eğrinin Altındaki Alana Dayalı Performans Analizi

Eğrinin Altındaki Alana Dayalı Performans Analizinde Genelleştirilmiş Doğrusal Model en iyi performansı göstermiştir ve ardından Konvolüsyonel Sinir Ağı, Naif Bayes gelmiştir.

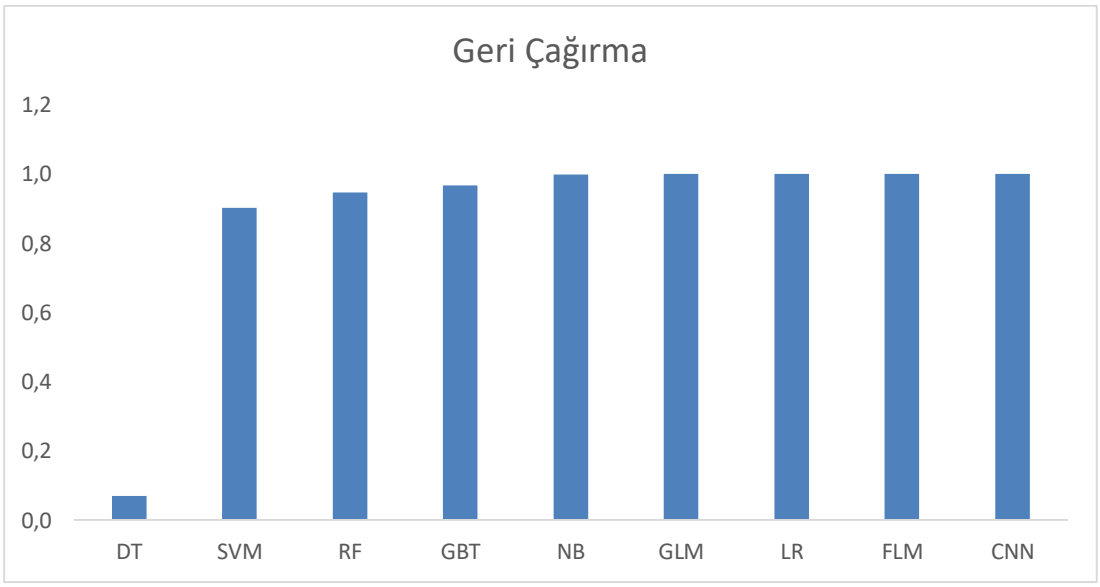
D. Hassasiyet (Precision) Analizi



Şekil 46. Hassasiyete Dayalı Performans Analizi

Hassasiyete Dayalı Performans Analizinde Genelleştirilmiş Doğrusal Model en iyi performansı göstermiştir ve ardından Naif Bayes, Hızlı Büyük Marj gelmiştir.

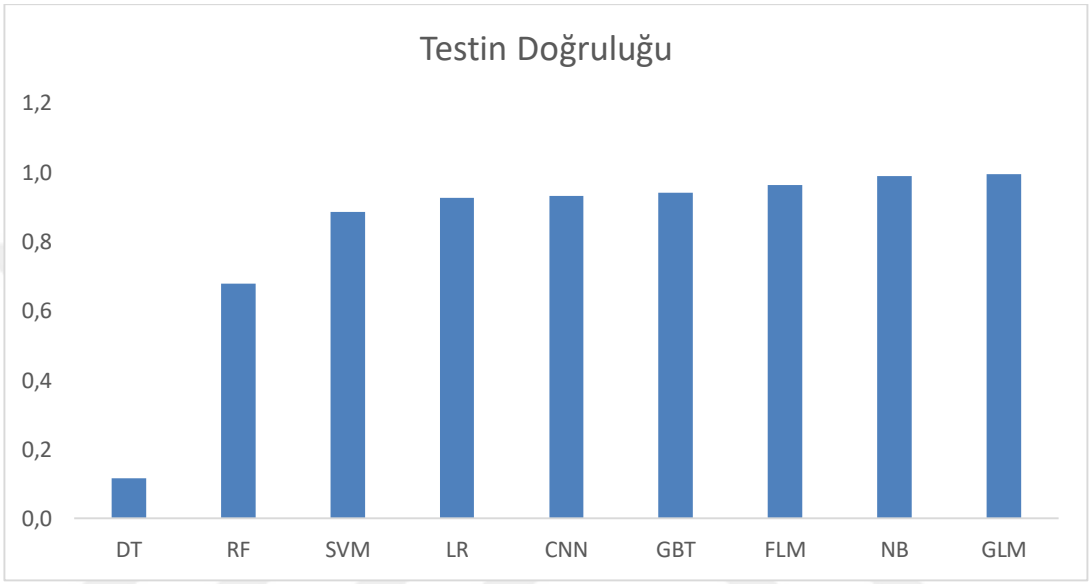
E. Geri Çağırma (Recall) Analizi



Şekil 47. Geri Çağırmaya Dayalı Performans Analizi

Geri Çağırmaya Dayalı Performans Analizinde Konvolüsyonel Sinir Ağı en iyi performansı göstermiştir ve ardından Hızlı Büyük Marj, Lojistik Regresyon gelmiştir.

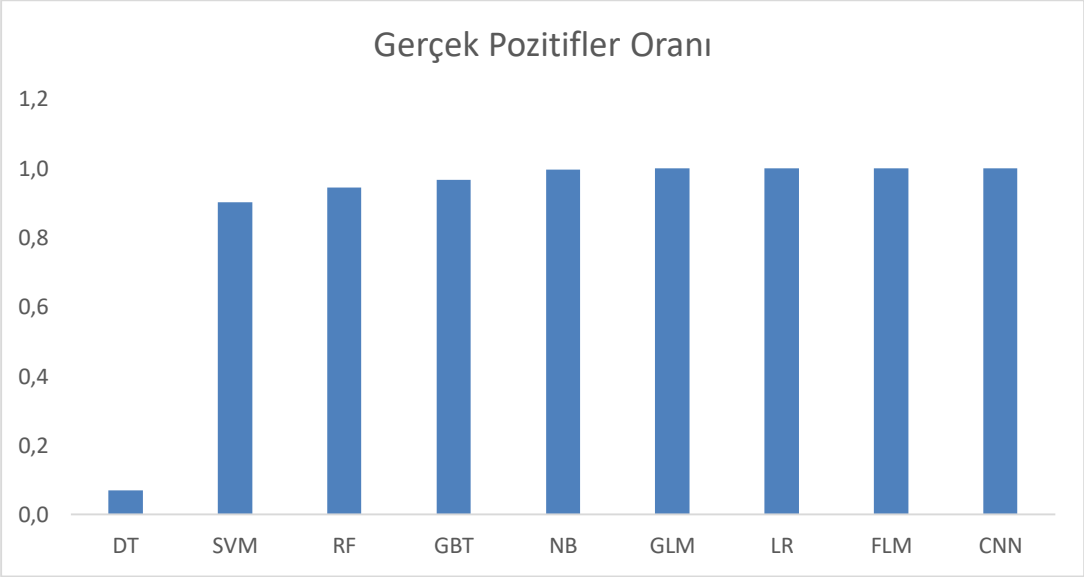
F. Testin Doğruluğu (F-measure) Analizi



Şekil 48. Testin Doğruluğuna Dayalı Performans Analizi

Testin Doğruluğuna Dayalı Performans Analizinde Genelleştirilmiş Doğrusal Model en iyi performansı göstermiştir ve ardından Naif Bayes, Hızlı Büyük Marj gelmiştir.

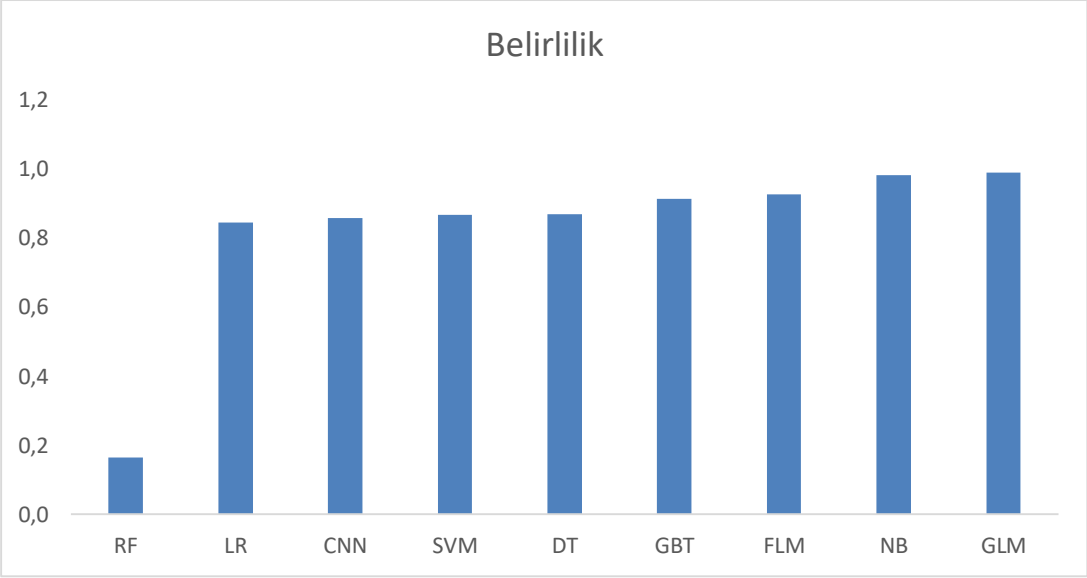
G. Gerçek Pozitifler Oranı (Sensitivity) Analizi



Şekil 49. Gerçek Pozitifler Oranına Dayalı Performans Analizi

Gerçek Pozitifler Oranına Dayalı Performans Analizinde Konvolüsyonel Sinir Ağı en iyi performansı göstermiştir ve ardından Hızlı Büyük Marj, Lojistik Regresyon gelmiştir.

H. Belirlilik (Specificity) Analizi



Şekil 50. Belirliliğe Dayalı Performans Analizi

Belirlilik bazında Genelleştirilmiş Doğrusal Model en iyi performansı göstermiştir ve ardından Naif Bayes, Hızlı Büyük Marj gelmiştir.

İ. Tartışma ve Öneriler

Hata Matrisi(Confusion Matrix – CM) hesaplamaları kullanılarak her bir algoritmanın farklı ölçütlerde sonuçları gösterilmiştir. Bu algoritmaların matematiksel gösterimlerinden ve kullanış biçimlerinden bahsedilmiştir. Kullanılan kodlar paylaşılmış, bu matematiksel gösterimlerin anlamları anlatılmıştır. Ayrıca neden bu algoritmaların seçildiği de, bu algoritmalar anlatılırken açıklanmıştır. Seçilen filtreleme yöntemleri, neden faydalı olacakları açıklanarak tanımlanmıştır. 9 adet algoritma kullanılmış, bunların 8 farklı hata matrisi formülü ayrı ayrı hesaplanarak, toplamda minimum 72 adet farklı sonuç kombinasyonu ortaya çıkarılmıştır. Genel anlamda bakıldığında kullanılan veri setine ve uygulanan ön işleme tekniklerine, ayrıca seçilen özelliklere ve veri setlerin bölünme yüzdelerine göre, en iyi performansı genelleştirilmiş doğrusal model vermiştir. Daha sonra Naif Bayes, Hızlı Büyük Marj, Konvolüsyonel Sinir Ağı ve Lojistik Regresyon gelmiştir. Burada algoritmanın sahip olduğu hız, bu tahminin sonucunu aşağı ya da yukarı yönde oynatabilmektedir. Karar Ağaçları ve Rastgele Orman algoritmaları en hızlı sonuç vermesine rağmen tahmin sonuç değerleri diğerlerinden oldukça düşük çıkmıştır. Buradan şu fikir çıkartılabilir: bir algoritmanın hızlı çalıştığı, iyi çalıştığı anlamına gelmeyebilir. Tabiki de makine öğrenmesi yöntemleri uygulanırken ve araştırılırken fark edilecektir ki, bu araştırmalar tamamiyle duruma özgü hesaplanır ve şu algoritma ötekine göre daha iyi çalışır diye bir genelleme yapmak çok doğru olmayacaktır. Bu sonuçlar, yapılan çalışmadan çıkan tek sonuçlar değildir. Bunların standart sapmaları, kazanımları (gain), toplam geçen süreleri, eğitim süreleri, çıkış (output) alma süreleri gibi kıstaslar da, ayrı birer çalışma konusu olabilir. Veri ön işleme yapılmayarak sonuçlar alınabilir, ve ardından veri ön işleme adımı yapılarak hesaplanabilir bu şekilde bu adımların (preprocessing) ne kadar önemli ve etkili olabileceğine dair bir çalışma da yapılabilir. Bu alanda kullanılan makine öğrenme yöntemleri ve bunların analizleri, bu analizlerin sonuçlarını kullanarak tahmin(prediction), sınıflandırma(classification), veri ön işleme(data preprocessing), özellik çıkarımı(feature extraction) alanlarında çalışmalar yapılabilir. Günlük işlenen ve kaydedilen veri miktarı her geçen gün artarken, bu veriyi analiz edecek ve

anlamlandırabilecek kişilere ve çalışmalara her zamankinden daha fazla ihtiyaç duyulmaktadır. Veri madenciliği ve makine öğrenme algoritmalarının kullanılabileceği bu veriler, birçok alanda fayda sağlayabilir örneğin maliyet azaltma, risk raporları, potansiyel kâr arttırımı, sağlık uygulamalarında iyileştirme, insan sağlığına fayda bulunma gibi birçok alanda kullanılabilir ve büyük faydalar sağlayabilir. Var olan algoritmalar incelenip eksikleri veya yanlışları varsa ortaya farklı çalışmalar çıkabilir ya da matematiksel bir düşünce varsa yeni bir algoritma ortaya atılabilir ki bu literatür açısından oldukça değerli bir durumdur. Bu tez okunarak gelecek çalışmalar için bir fikre sahip olunabilir ya da fikirlerin değişmesini, güncellenmesini sağlayacak tecrübeler ortaya çıkabilir. Bu testleri yapabilecek daha başka yazılımlar denenebilir veya burada görülen bilgileri test etmeye yarayacak başka sınıflandırma yöntemleri, kümeleme yöntemleri ve veri ön işleme yöntemleri karşılaştırılabilir. Başka veri setleri aynı yöntemler kullanılarak karşılaştırma yapılabilir ve bunlar bir araya getirilerek yeni sonuçlar ortaya atılabilir. Bu tezde bahsedilen algoritmalar dışında daha başka birçok algoritma, yöntem ve fikir bulunduğundan; bunlar kapasitesi fazla ve eklentiler ile genişleyebilen yazılımlarda uygulayarak çeşitli veriler üzerinden birçok farklı çıkarımlar elde edilebilir.

KAYNAKLAR

Abadi M., vd. (2016). "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." arXiv:1603.04467, 2016

Batra S. , Sachdeva S. (2016). "Organizing standardized electronic healthcare records data for mining. "

Bishop C.M. (1995). "Neural Networks for Pattern Recognition." Oxford University Press, First Edition, 1995

Breiman L. (2001). "Random Forests." Machine Learning, vol. 45, no. 1, sayfa 5-32, 2001

Bolboaca D.S. , Jantschi L., Sestraş A.F. , Sestraş R.E. , Pamfil D.C. (2011). "Pearson-Fisher Chi-Square Statistic Revisited." ISSN 2078-2489, sayfa 528-545, 2011

Bottou L., Chapelle O. , Decoste D. , Weston J. (2007). "Scaling Learning Algorithms towards AI Large-Scale Kernel Machines." MIT Press,2007

Cao R. , Yu Z. , Marbach T. , Li Y. , Wang G., Liu X. (2018). "Load Prediction for Data Centers Based on Database Service." 42nd International Conference on Computer Software & Applications, 2018

Chapelle O. , Vapnik V. , Bousquet O. , Mukherjee S. (2002). "Choosing Multiple Parameters for Support Vector Machines." Machine Learning, vol. 46, no.1, sayfa 131-159,2002

Chen T. , Guestrin C. (2016). "A scalable tree boosting system." KDD, sayfa 785-794, 2016

Chetan A. , Sultane A. , Bhalerao M.V. , Bonde S. (2017). "Character Recognition Based on Skeletonization: A Survey." Int. J. Adv. Res. 5(6), 1503-1519, June 2017

Ciresan D. , Meier U. , Gambardella L.M. , Schmidhuber J. (2010). "Deep Big Simple Neural Nets Excel on Handwritten Digit Recognition." Neural Computation, Volume 22, Number 12, DOI: 10.1162/NECO_a_00052, December 2010

David S. B. , Shwartz S.S. (2014). "Understanding Machine Learning: From Theory to Algorithms. "

Dietterich T.G., Bakiri G. (1995). "Solving Multiclass Learning Problems via Error-Correcting Output Codes." J. Artificial Intelligence Research, vol. 2, sayfa 263-286, 1995

Dorian P. (2006). "Data Preparation for Data Mining."

Duggal R. , S. Shukla, S. Chandra, B. Shukla ve S.K.Khatri (2016). "Impact of selected pre-processing techniques on prediction of risk of early readmission for diabetic patients in India."

Freund Y. , Schapire R. (1999). "A short introduction to boosting." Journ. of Jap. Soc. For Artificial Intelligence, vol. 14, no.5, sayfa 771-780, 1999

Friedman J. , Hastie T. , Tibshirani R. (2000). "Additive logistic regression: a statistical view of boosting." Annals of statistics, vol. 28, no. 2, sayfa 337-407, 2000

Friedman J. (2001). "Greedy function approximation: a gradient boosting machine." Annals of statistics, sayfa 1189-1232, 2001

Gilad-Bachrach R. , Navot A. , Tishby N. (2004). "Margin Based Feature Selection-Theory and Algorithms." Proc. 21st International Conference of Machine Learning, sayfa 43-50, 2004

Glorot X. , Bengio Y. (2010). "Understanding the difficulty of training feedforward neural networks." 13th International Conference on Artificial Intelligence and Statistics (AISTATS) Chi Laguna Resort, Sardinia, Italy. Volume 9 of JMLR: W&CP 9, 2010

Guyon I. , Weston J. , Barnhill S. , Vapnik V. (2002). "Gene Selection for Cancer Classification using Support Vector Machines." Machine Learning, vol. 36, nos. 1-3, sayfa 389-422, 2002

Guyon I. , Elisseeff A. (2003). "An Introduction to Variable and Feature Selection." J. Machine Learning Research, vol.3, sayfa 1157-1182, 2003

Hameed S.S. , Muhammad F.F. , Hassani R. , Saeed F. (2018). "Gene Selection and Classification in Microarray Datasets using a Hybrid Approach of PCC-BPSO/GA with Multi Classifiers." Journal of Computer Science, 2018

Hebb D.O. (1949). "The Organization of Behavior." Wiley, 1949

Hinton G.E. , Salakhutdinov R.R. (2006). "Reducing the dimensionality of data with neural networks." Science, 313(5786):504-507,2006

Hinton G.E. , Srivastava N. , Krizhevsky A. , Sutskever I. , Salakhutdinov R.R. (2012). "Improving neural networks by preventing co-adaptation of feature detectors." arXiv:1207.0580v1 [cs.NE] , July 2012

Hinton G., Deng L., Yu D. , Dahl G. , Mohamed A. , Jaitly N. , Senior A. , Vanhoucke V. , Nguyen P. , Sainath T, Kingsbury B. (2012). "Deep Neural Networks for Acoustic Modeling in Speech Recognition."

Hilario M., Kalousis A. (2008). "Approaches to Dimensionality Reduction in Proteomic Biomarker Studies." Briefings in Bioinformatics, vol.9,no.2, sayfa 102-118,2008

Hopfield J. J. (1982). "Neural networks and physical system with emergent collective computational abilities." Institute for Cognitive Science, C-015, University of California, USA, Nature Vol 323-329, October 1986

Hotelling H. (1933). "Analysis of a complex of statistical variables into principal components." Journal of Educational Psychology, vol. 24, sayfa 417-441, 1933

Jiang J. , Jiang J. , Cui B. , Zhang C. (2017). "TencentBoost: A Gradient Boosting Tree System with Parameter Server." IEEE 33rd International Conference on Data Engineering, 2375-026X/17, DOI 10.1109/ICDE.2017.87,sayfa 281-284, 2017

Jidiga G.R. , Sammulal P. (2013). "Anomaly Detection Using Generic Machine Learning Approach with a Case Study of Awareness." International Journal of Modern Engineering Research (IJMER) ISSN:2249-6645, Vol.3, Issue.2, sayfa 1245-1252 March-April 2013

Kara S. , Direngali F. (2007). "A system to diagnose atherosclerosis via wavelet transforms principal component analysis and artificial neural networks." Expert Systems with Applications, vol. 32, sayfa 632-640, 2007

Kira K. , Rendell L.A. (1992). "A Practical Approach to Feature Selection." Proc. Ninth International Conference of Machine Learning, sayfa 249-256, 1992

Kohavi R. , John G.H. (1997). "Wrappers for Feature Subset Selection." Artificial Intelligence, vol.97, nos. ½ sayfa 273-324, 1997

Koller D. , Sahami M. (1996). "Toward Optimal Feature Selection." Proc. 13th International Conference of Machine Learning, sayfa 284-292,1996

Krizhevsky A. , Sutskever I. , Hinton G.E. (2012). "ImageNet Classification with Deep Convolutional Neural Networks."

Lal T.N. , Chapelle O. , Weston J. , Elisseeff A. (2006). "Embedded Methods." Feature Extraction Foundations and Applications, sayfa 137-165, Springer-Verlag, 2006

Laurene F. (1994). "Fundamentals of Neural Networks Architectures Algorithms and Applications." Prentice-Hall, ISBN 0133341860, 1994

Liaw A. , Wiener M. (2002). “Classification and regression by randomForest.” R News, vol. 2, no. 3, sayfa 1820, 2002

Lin C.J. , Weng R.C. , Keerthi S.S. (2008). "Trust region Newton method for logistic regression.” Vol. 9, sayfa 627-650, ISSN 1532-4435, 2008

Masters T. (1995). "Neural, Novel&Hybrid Algorithms for Time Series Prediction.” Journal of the American Statistical Association, Vol.94, No. 445, p.347

McCulloch W.S. , Pitts W.H. (1943). “A Logical Calculus of the ideas immanent in nervous activity.” The Bulletin of Mathematical Biophysics, 5(4): sayfa 115-133, 1943

Minsky L., Papert S.A. (1988). “Perceptrons.” MIT Press, 1988

Naik H. , Kanikar P. (2019). “Credit card Fraud Detection based on Machine Learning Algorithms.” International Journal of Computer Applications(0975-8887) Volume 182 – No. 44, March 2019

NG A. Y. (2004). "Feature Selection, L1 vs L2 Regularization, and Rotational Invariance.” Proc. 21st International Conference of Machine Learning, sayfa 78-86,2004

Pudil P. , Novovicova J. (1998). “Novel Methods for Subset Selection with Respect to Problem Knowledge.” IEEE Intelligent Systems, vol. 13, no. 2, sayfa 66-74, March 1998

Raina R. , Madhavan A. , NG A. Y. (2009). “Large-scale Deep Unsupervised Learning using Graphics Processors.” Proceedings of the 26th International Conference on Machine Learning, Montreal, Canada 2009

Remes V. , Haindly M. (2015). “Classification of Breast Density in X-ray Mammography.” ISBN: 978-1-4673-8457-5/1, IEEE, 2015

Rey R.F. (1983). “Engineering and Operations in the Bell System.”

Rosenblatt F. (1958). “The Perceptron: A probabilistic model for information storage and organization in the brain.” Vol.66 No. 6, 1958

Rumelhart D.E. , Hinton G.E. , Williams R.J. (1986). “Parallel distributed processing: Explorations in the microstructure of cognition.” Vol. 1. Chapter Learning Internal Representations by Error Propagation, pages 318-362.MIT Press, Cambridge, ISBN 0-262-68053-X, 1986

Rumelhart D.E. , Hinton G.E. , Williams R.J. (1986). “Learning representations by back-propagating errors.”, Nature 323, p533-536

Russel S.J. , Norvig P. (1995). “Artificial Intelligence: A Modern Approach.” PrenticeHall Inc, First Edition, 1995

Samuel A.L. (1959). “Some studies in Machine Learning using the Game of Checkers.”

Sejnowski T. J. , Rosenberg C.R. (1986). “NETtalk: a parallel network that learns to read aloud.”

Smelser N. J. , Baltes P.B. (2001). “International Encyclopedia of Social & Behavioral Sciences.” First Edition, ISBN: 9780080548050, Pergamon 2001

Sun Y. , Todorovic S. , Li Y. , Wu D. (2005). “Unifying Error-Correcting and Output-Code AdaBoost through the Margin Concept.” Proc. 22nd International Conference of Machine Learning, sayfa 872-879, 2005

Taigman Y. , Yang M., Ranzato M. , Wolf L. (2014). “DeepFace: Closing the Gap to Human-Level Performance in Face Verification.” CVPR 2014

Turing A.M. (1936). “On Computable Numbers, with an application to the Entscheidungs problem.”

Turing A.M. (1950). “Computing Machinery and Intelligence.” Mind 49: 433-460, 1950

Vant V.L. (2002). “Gene Expresion Profiling Predicts Clinical Outcome of Breast Cancer.” Nature, vol.415 sayfa530-536,2002

Wang Y. (2005). "Gene-Expression Profiles to Predict Distant Metastasis of Lymph-Node Negative Primary Breast Cancer." Lancet, vol. 365, sayfa 671-679, 2005

Wei J. , vd. (2015). "Managed communication and consistency for fast data-parallel iterative analytics." SoCC, sayfa 381-394, 2015

Werbos P.J. (1994). "The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting." A Wiley Interscience Publication, Wiley, 1994

Weston J. , Mukherjee S. , Chapelle O. , Pontil M., Poggio T. , Vapnik V. (2001). "Feature Selection for SVMs." Proc. 13th Advances in Neural Information Processing Systems, sayfa 668-674, 2001

Zhang X. , Gonnot T. , Saniie J. (2017). "Real-Time Face Detection and Recognition in Complex Background." Journal of Signal and Information Processing, ISSN: 2159-4481 8, sayfa 99-112, 2017

Zhu J. , Rosset S. , Hastie T. , Tibshirani R. (2004). "1-Norm Support Vector Machines." Proc. 16th Advances in Neural Information Processing Systems, 2004

Internet Kaynakları:

Url-1: Binnaka Asu, (11 Mart 2018) "AWS Üzerinde Makine Öğrenimi" , Amazon (Erişim Tarihi: 13/09/2019) <https://aws.amazon.com/tr/machine-learning/>

Url-2: Kaiming He, (10 Aralık 2015) "Deep Residual Learning for Image Recognition", Arxiv (Erişim Tarihi: 12/09/2019) <https://arxiv.org/abs/1512.03385>

Url-3: B.J. Copeland, (21 Kasım 2018) "MYCIN, Artificial Intelligence Program", Britannica (Erişim Tarihi: 14/09/2019) <https://www.britannica.com/technology/MYCIN>

Url-4:Doug Gross, (04 Ekim 2011), CNN (Erişim Tarihi: 12 Eylül 2019). <https://edition.cnn.com/2011/10/04/tech/mobile/siri-iphone-4s-skynet/index.html>

Url-5: Yevgeniy Sverdlik, (08 Eylül 2016), DataCenterKnowledge (Erişim Tarihi: 19/09/ 2019) <http://www.datacenterknowledge.com/archives/2016/09/08/delta-data-center-outage-cost-us-150m>

Url-6: Guru Ecseg, (04 Mart 2014), ComputerScience (Erişim Tarihi: 12/09/2019) <https://www.computerscience.gcse.guru/theory/von-neumann-architecture>

Url-7: Yaniv Leviathan, (08 Mayıs 2018) GoogleBlog (Erişim Tarihi: 12/09/2019) <https://ai.googleblog.com/2018/05/duplex-ai-system-for-natural-conversation.html>

Url-8: Jeff Dean, (03 Şubat 2014) GoogleBrainTeam (Erişim Tarihi: 13/09/2019) <https://ai.google/research/teams/brain/>

Url-9: Chung-Jen Tan, (01 Şubat 2011) IBM, (Erişim Tarihi: 12/09/2019) <https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/>

Url-10: Steven Levy, (16 Ocak 2015) Medium (Erişim Tarihi: 16/09/2019) <https://medium.com/backchannel/google-search-will-be-your-next-brain-5207c26e4523#.b3x9b7ods>

Url-11: Paul Bommarito, (11 Mayıs 2016) Nvidia (Erişim Tarihi: 14/09/2019) <https://www.nvidia.com/en-us/data-center/tesla-v100/>

Url-12: Andrew Pollack, (05 Haziran 1992) NYTimes (Erişim Tarihi: 12/09/2019) <https://www.nytimes.com/1992/06/05/business/fifth-generation-became-japan-s-lost-generation.html>
Url-13: Greg Brockman, (11 Aralık 2015), OpenAI (Erişim Tarihi: 13/09/2019) <https://openai.com>
Url-14: Jennifer D. Chee, (04 Mayıs 2015), ResearchGate (Erişim Tarihi: 21/09/2019) <https://www.researchgate.net/publication/277324930_Pearson's_Product-Moment_Correlation_Sample_Analysis>
Url-15: Robert Nisbet, (14 Mart 2018), ScienceDirect (Erişim Tarihi: 14/09/2019) <https://www.sciencedirect.com/topics/computer-science/data-filtering>
Url-16: Dennis G. Jerz, (30 Ocak 2006), SetonHill (Erişim Tarihi: 14/09/2019) <https://jerz.setonhill.edu/if/canon/eliza.htm>
Url-17: Apache Communtiy, (10 Ocak 2012), Apache (Erişim Tarihi: 26/09/2019) <http://spark.apache.org/mllib/>
Url-18: Josh Lewis, (05 Ekim 2000) Swarthmore (Erişim Tarihi: (14/09/2019) <https://www.cs.swarthmore.edu/~eroberts/cs91/projects/ethics-of-ai/sec1_2.html>
Url-19: Jesus Rordiguez, (16 Eylül 2019) TowardsDataScience (Erişim Tarihi: 21/09/2019) <https://towardsdatascience.com/deepmind-quietly-open-sourced-three-new-impressive-reinforcement-learning-frameworks-f99443910b16>
Url-20: Heet Sankesera, (23 Ocak 2019), TowardsDataScience (Erişim Tarihi: 14/09/2019) <https://towardsdatascience.com/u-net-b229b32b4a71>
Url-21: Nagesh Singh Chauhan, (11 Mart 2019), TowardsDataScience (Erişim Tarihi: 23/09/2019) <https://towardsdatascience.com/real-world-implementation-of-logistic-regression-5136cefb8125>
Url-22: Rory Carroll, (21 Kasım 2015) TheGuardian (Erişim Tarihi: 13/09/2019) <https://www.theguardian.com/technology/2015/nov/21/amazon-echo-alexa-home-robot-privacy-cloud>
Url-23: Adam Gabbatt, (17 Şubat 2011) TheGuardian (Erişim Tarihi: 14/09/2019) <https://www.theguardian.com/technology/2011/feb/17/ibm-computer-watson-wins-jeopardy>
Url-24: Tom Warren, (02 Nisan 2014) TheVerge (Erişim Tarihi: 12/09/2019) <https://www.theverge.com/2014/4/2/5570866/cortana-windows-phone-8-1-digital-assistant>
Url-25: Kevin Heslin, (09 Eylül 2014), UptimeEnstitute (Erişim Tarihi: 19/09/2019) <https://journal.uptimeinstitute.com/data-center-outages-incidents-industry-transparency/
Url-26: Breaking Research Team, (26 Kasım 2012) UToronto (Erişim Tarihi: 16/09/2019) <https://www.utoronto.ca/news/leading-breakthroughs-speech-recognition-software-microsoft-google-ibm >

EKLER

EK-A

A.1: Normalizasyon Python Kodu

```
import pandas pd
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

from sklearn.linear_model import Ridge
X_train, X_test, y_train, y_test = train_test_split(X_crime, y_crime,
                                        random_state = 0)

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

linridge = Ridge(alpha=20.0).fit(X_train_scaled, y_train)
```

EK-B

B.1: Naif Bayes Python Kodu

```
from csv import reader
from math import sqrt
from math import exp
from math import pi

# Load a CSV file
def load_csv(filename):
	dataset = list()
	with open(filename, 'r') as file:
		csv_reader = reader(file)
		for row in csv_reader:
			if not row:
				continue
			dataset.append(row)
	return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
	for row in dataset:
		row[column] = float(row[column].strip())

# Convert string column to integer
def str_column_to_int(dataset, column):
	class_values = [row[column] for row in dataset]
	unique = set(class_values)
	lookup = dict()
	for i, value in enumerate(unique):
		lookup[value] = i
```

```
            print('[%s] => %d' % (value, i))
    for row in dataset:
            row[column] = lookup[row[column]]
    return lookup

# Split the dataset by class values, returns a dictionary
def separate_by_class(dataset):
    separated = dict()
    for i in range(len(dataset)):
            vector = dataset[i]
            class_value = vector[-1]
            if (class_value not in separated):
                    separated[class_value] = list()
            separated[class_value].append(vector)
    return separated

# Calculate the mean of a list of numbers
def mean(numbers):
    return sum(numbers)/float(len(numbers))

# Calculate the standard deviation of a list of numbers
def stdev(numbers):
    avg = mean(numbers)
    variance = sum([(x-avg)**2 for x in numbers]) / float(len(numbers)-1)
    return sqrt(variance)

# Calculate the mean, stdev and count for each column in a dataset
def summarize_dataset(dataset):
    summaries = [(mean(column), stdev(column), len(column)) for column in
    zip(*dataset)]
    del(summaries[-1])
    return summaries

# Split dataset by class then calculate statistics for each row
def summarize_by_class(dataset):
    separated = separate_by_class(dataset)
    summaries = dict()
    for class_value, rows in separated.items():
            summaries[class_value] = summarize_dataset(rows)
    return summaries

# Calculate the Gaussian probability distribution function for x
def calculate_probability(x, mean, stdev):
    exponent = exp(-((x-mean)**2 / (2 * stdev**2 )))
    return (1 / (sqrt(2 * pi) * stdev)) * exponent

# Calculate the probabilities of predicting each class for a given row
def calculate_class_probabilities(summaries, row):
    total_rows = sum([summaries[label][0][2] for label in summaries])
    probabilities = dict()
```

```
    for class_value, class_summaries in summaries.items():
        probabilities[class_value] = summaries[class_value][0][2]/float(total_rows)
        for i in range(len(class_summaries)):
            mean, stdev, _ = class_summaries[i]
            probabilities[class_value] *= calculate_probability(row[i], mean,
    stdev)
    return probabilities

# Predict the class for a given row
def predict(summaries, row):
    probabilities = calculate_class_probabilities(summaries, row)
    best_label, best_prob = None, -1
    for class_value, probability in probabilities.items():
        if best_label is None or probability > best_prob:
            best_prob = probability
            best_label = class_value
    return best_label
```

EK-C

C.1: Genelleştirilmiş Doğrusal Model Python Kodu

```
import numpy as np
import scipy.stats as sts
import patsy as pt

from .utils import (check_types, check_commensurate, check_intercept,
        check_offset, check_sample_weights, has_converged,
        default_X_names, default_y_name)
from .families import Gaussian


class GLM:

  def __init__(self, family, alpha=0.0):
    self.family = family
    self.alpha = alpha
    self.formula = None
    self.X_info = None
    self.X_names = None
    self.y_name = None
    self.coef_ = None
    self.deviance_ = None
    self.n = None
    self.p = None
    self.information_matrix_ = None

  def fit(self, X, y=None, formula=None, *,
      X_names=None,
      y_name=None,
      **kwargs):
```

```
    check_types(X, y, formula)
    if formula:
        self.formula = formula
        y_array, X_array = pt.dmatrices(formula, X)
        self.X_info = X_array.design_info
        self.X_names = X_array.design_info.column_names
        self.y_name = y_array.design_info.term_names[0]
        y_array = y_array.squeeze()
        return self._fit(X_array, y_array, **kwargs)
    else:
        if X_names:
            self.X_names = X_names
        else:
            self.X_names = default_X_names(X)
        if y_name:
            self.y_name = y_names
        else:
            self.y_name = default_y_name()
        return self._fit(X, y, **kwargs)

def _fit(self, X, y, *,
         warm_start=None,
         offset=None,
         sample_weights=None,
         max_iter=100,
         tol=0.1**5):

    check_commensurate(X, y)
    check_intercept(X)
    if warm_start is None:
        initial_intercept = np.mean(y)
        warm_start = np.zeros(X.shape[1])
        warm_start[0] = initial_intercept
    coef = warm_start
    if offset is None:
        offset = np.zeros(X.shape[0])
    check_offset(y, offset)
    if sample_weights is None:
        sample_weights = np.ones(X.shape[0])
    check_sample_weights(y, sample_weights)

    family = self.family
    penalized_deviance = np.inf
    is_converged = False
    n_iter = 0
    while n_iter < max_iter and not is_converged:
        nu = np.dot(X, coef) + offset
        mu = family.inv_link(nu)
        dmu = family.d_inv_link(nu, mu)
```

```
        var = family.variance(mu)
        dbeta = self._compute_dbeta(X, y, mu, dmu, var, sample_weights)
        ddbeta = self._compute_ddbeta(X, dmu, var, sample_weights)
        if self._is_regularized():
            dbeta = dbeta + self._d_penalty(coef)
            ddbeta = self._dd_penalty(ddbeta, X)
        coef = coef - np.linalg.solve(ddbeta, dbeta)
        penalized_deviance_previous = penalized_deviance
        penalized_deviance = family.penalized_deviance(
            y, mu, self.alpha, coef)
        is_converged = has_converged(
            penalized_deviance, penalized_deviance_previous, tol)
        n_iter += 1

    self.coef_ = coef
    self.deviance_ = family.deviance(y, mu)
    self.n = np.sum(sample_weights)
    self.p = X.shape[1]
    self.information_matrix_ = self._compute_ddbeta(X, dmu, var, sample_weights)
    return self

def predict(self, X, offset=None):

    if not self._is_fit():
        raise ValueError(
            "Model is not fit, and cannot be used to make predictions.")
    if self.formula:
        rhs_formula = '+'.join(self.X_info.term_names[1:])
        X = pt.dmatrix(rhs_formula, X)
    if offset is None:
        return self.family.inv_link(np.dot(X, self.coef_))
    else:
        return self.family.inv_link(np.dot(X, self.coef_) + offset)

def score(self, X, y):

    return self.family.deviance(y, self.predict(X))

@property
def dispersion_(self):

    if not self._is_fit():
        raise ValueError("Dispersion parameter can only be estimated for a"
                         "fit model.")
    if self.family.has_dispersion:
        return self.deviance_ / (self.n - self.p)
    else:
        return np.ones(shape=self.deviance_.shape)

@property
```

```
def coef_covariance_matrix_(self):
    if not self._is_fit():
        raise ValueError("Parameter covariances can only be estimated for a"
                         "fit model.")
    return self.dispersion_ * np.linalg.inv(self.information_matrix_)

@property
def coef_standard_error_(self):
    return np.sqrt(np.diag(self.coef_covariance_matrix_))

@property
def p_values_(self):

    if self.alpha != 0:
        raise ValueError("P-values are not available for "
                         "regularized models.")
    p_values = []
    null_dist = sts.norm(loc=0.0, scale=1.0)
    for coef, std_err in zip(self.coef_, self.coef_standard_error_):
        z = abs(coef) / std_err
        p_value = null_dist.cdf(-z) + (1 - null_dist.cdf(z))
        p_values.append(p_value)
    return np.asarray(p_values)

def summary(self):

    variable_names = self.X_names
    parameter_estimates = self.coef_
    standard_errors = self.coef_standard_error_
    header_string = "{:<10} {:>20} {:>15}".format(
        "Name", "Parameter Estimate", "Standard Error")
    print(f"{self.family.__class__.__name__} GLM Model Summary.")
    print('='*len(header_string))
    print(header_string)
    print('-'*len(header_string))
    format_string = "{:<20} {:>10.2f} {:>15.2f}"
    for name, est, se in zip(variable_names, parameter_estimates, standard_errors):
        print(format_string.format(name, est, se))

def clone(self):
    return self.__class__(self.family, self.alpha)

def _is_fit(self):
    return self.coef_ is not None

def _is_regularized(self):
    return self.alpha > 0.0

def _compute_dbeta(self, X, y, mu, dmu, var, sample_weights):
    working_residuals = sample_weights * (y - mu) * (dmu / var)
```

```
        return - np.sum(X * working_residuals.reshape(-1, 1), axis=0)

    def _compute_ddbeta(self, X, dmu, var, sample_weights):
        working_h_weights = (sample_weights * dmu**2 / var).reshape(-1, 1)
        return np.dot(X.T, X * working_h_weights)

    def _d_penalty(self, coef):
        dbeta_penalty = coef.copy()
        dbeta_penalty[0] = 0.0
        return dbeta_penalty

    def _dd_penalty(self, ddbeta, X):
        diag_idxs = list(range(1, X.shape[1]))
        ddbeta[diag_idxs, diag_idxs] += self.alpha
        return ddbeta
```

EK-D

D.1: Lojistik Regresyon C++ Kodu

```
#include <iostream>
#include <fstream>
#include <ctime>
#include <cmath>
#include "logistic_regression.h"
using namespace std;

LogisticRegression::LogisticRegression(const char* configure)
{
int status=CommonTool::load_configure(configure,this->conf_dict);
if(status<0)
{
cout<<"load configure file:"<<configure<<" faild."<<endl;
return;
}
//new (&dummyConvertor)DummyConvertor(conf_dict["dummy_conf"].c_str());
dummyConvertor=DummyConvertor(conf_dict["dummy_conf"].c_str());
this->p_data_stream=new istringstream();
this->cursor=0;
this->lbfgs_iterations=0;
```

```
this->save_intermediate_peroid=10;
}
lbfgsfloatval_t LogisticRegression::evalute(const lbfgsfloatval_t *w,lbfgsfloatval_t *g,const int n,const lbfgsfloatval_t step)
{
cout<<"evalute start"<<endl;
if(g!=0)
memset(g,0,sizeof(g)*n);
// new (&(this->data_stream)) istringstream(this->data_buffer);
this->prepare_read();
vector<string> field_vec;
CommonTool::split(this->conf_dict[string("filter:field_name_vec")],',',field_vec);
string line;
vector<string> record;
set<string> fields;
lbfgsfloatval_t NLL=0.0;
int line_cnt=0;
time_t t=time(0);
while(get_line(line))
{
line_cnt++;
if(line_cnt%1000000==0)
{
t=time(0);
cout<<"progress:"<<line_cnt<<" time:"<<asctime(localtime(&t))<<endl;
}
CommonTool::split(line,'\t',record);
/*
int yi=atoi(record.back().c_str());
LogisticRegression::convert_record(record,fields,field_vec);
lbfgsfloatval_t ti=0.0;
for(set<string>::const_iterator iter=fields.begin();iter!=fields.end();iter++)
{
```

```
int i=this->dummyConvertor.convert(*iter);
// cout<<"index="<<i<<" record:"<<line<<endl;
// cout.flush();
if(i>=n)
{
cout<<"Error index:"<<i<<" for item:"<<*iter<<endl;
}
ti+=w[i];
}
lbfgsfloatval_t ui=CommonTool::sigmod(ti);
lbfgsfloatval_t gradient=ui-yi;
for(set<string>::const_iterator iter=fields.begin();iter!=fields.end();iter++)
{
int i=this->dummyConvertor.convert(*iter);
g[i]+=gradient;
}
*/
int yi=atoi(record.front().c_str());
lbfgsfloatval_t ui=predict(record,w);
lbfgsfloatval_t gradient=ui-yi;
if(g!=0)
{
for(int j=1;j<record.size();j++)
{
int k=this->dummyConvertor.convert(record[j]);
g[k]+=gradient;
}
}
NLL+=-(yi*log(ui)+(1-yi)*log(1-ui));
}
this->finish_read();
this->lbfgs_iterations++;
if(this->lbfgs_iterations%this->save_intermediate_peroid==0 && this-
```

```
>intermediate_file_prefix.size()>0)
{
string parameter_values;
this->get_parameter_values(parameter_values);
ostringstream sout;
sout<<this->intermediate_file_prefix<<"."<<this->lbfgs_iterations;
ofstream fout(sout.str().c_str());
fout<<parameter_values;
fout.close();
}
cout<<"end evalute,NLL="<<NLL<<endl;
return NLL;
}
int LogisticRegression::optimize(const string& intermediate_file_prefix,string&
final_para_results)
{
this->lbfgs_iterations=0;
this->intermediate_file_prefix=intermediate_file_prefix;
int n=dummyConvertor.get_length();
this->parameters=lbfgs_malloc(n);
if(this->parameters==0)
{
cout<<"malloc memery for parameters faild."<<endl;
return -1;
}
double rmax=(double)RAND_MAX;
for(int i=0;i<n;i++)
{
this->parameters[i]=(rand()/rmax)*2-1;
}
/// end of random initialize
lbfgsfloatval_t NLL=-1.0;
time_t t=time(0);
```

```
cout<<"begin LBFGS, time:"<<asctime(localtime(&t))<<endl;
int status=lbfgs(n,this->parameters,&NLL,_evalute,0,this,0);
t=time(0);
cout<<"end LBFGS. NLL="<<NLL<<" time:"<<asctime(localtime(&t))<<endl;
get_parameter_values(final_para_results);
lbfgs_free(this->parameters);
return status;
}
int LogisticRegression::get_parameter_values(string &result)const
{
int n=this->dummyConvertor.get_length();
ostringstream sout;
for(int i=0;i<n;i++)
sout<<this->parameters[i]<<endl;
result.assign(sout.str());
return n;
}
int LogisticRegression::load_parameter_from_file(const char* filename)
{
ifstream fin(filename);
if(!fin)
{
cout<<"Error, open "<<filename<<" failed."<<endl;
return -1;
}
vector<string> lines;
string line;
while(getline(fin,line))
{
lines.push_back(line);
}
this->parameters=new lbfgsfloatval_t[lines.size()];
if(this->parameters==0)
```

```
{
cout<<"Error, allocate memery faild."<<endl;
return -1;
}
for(size_t i=0;i<lines.size();i++)
{
this->parameters[i]=atof(lines[i].c_str());
}
return lines.size();
}
double LogisticRegression::predict(const char *parameters_file,const char
*test_file,const char *output)
{
if(this->load_parameter_from_file(parameters_file)<0)
return -1.0;
ifstream fin(test_file);
if(!fin)
{
cout<<"Error, open test file:"<<test_file<<" failed."<<endl;
return -1.0;
}
ofstream fout(output);
if(!fout)
{
cout<<"Error,open output file:"<<output<<" failed."<<endl;
return -1.0;
}
string line;
vector<string> record;
double NLL=0.0;
while(getline(fin,line))
{
CommonTool::split(line,'\t',record);
```

```
double ui=this->predict(record,this->parameters);
int yi=atoi(record.front().c_str());
NLL+=-(yi*log(ui)+(1-yi)*log((1-ui)));
fout<<ui<<","<<record.front()<<endl;
}
fin.close();
fout.close();
if(this->parameters!=0)
delete this->parameters;
return NLL;
}
double LogisticRegression::predict(const vector<string>& record, const
lbfgsfloatval_t *w)const
{
double ti=0.0;
for(int j=1;j<record.size();j++)
{
int k=this->dummyConvertor.convert(record[j]);
ti+=w[k];
}
double ui=CommonTool::sigmod(ti);
return ui;
}
int LogisticRegression::init_data_buffer(const char* train_file)
{
ifstream fin(train_file);
if(!fin)
{
cout<<"open file:"<<train_file<<" faild."<<endl;
return -1;
}
fin.seekg(0,fin.end);
unsigned int file_size=fin.tellg();
```

```
fin.seekg(0,fin.beg);
/* char *buffer=new char[file_size+1];
if(buffer==0)
{
cout<<"alloct memory faild, file size="<<file_size<<endl;
return -1;
}
*/
this->data_buffer.resize(file_size);
fin.read(&(this->data_buffer[0]),file_size);
fin.close();
cout<<"read file,size="<<this->data_buffer.size()<<endl;
return 0;
}
int LogisticRegression::prepare_read()
{
// this->data_stream.seekg(0);
// this->p_data_stream=new (this->p_data_stream)istringstream(this->data_buffer);
this->cursor=0;
return 0;
}
int LogisticRegression::finish_read()
{
// this->p_data_stream->~istringstream();
// this->data_stream.close();
return 0;
}
/*
istream& LogisticRegression::get_line(string &line)
{
return getline(*(this->p_data_stream),line);
}
*/
```

```
bool LogisticRegression::get_line(string &line)
{
if(this->cursor>=this->data_buffer.size())
return false;
line.clear();
while(this->cursor<this->data_buffer.size() && this->data_buffer[this->cursor]!='\n')
{
line.push_back(this->data_buffer[this->cursor]);
this->cursor++;
}
this->cursor++;
return true;
}
LogisticRegression::~LogisticRegression()
{
if(p_data_stream!=0)
delete p_data_stream;
}
int LogisticRegression::convert_record(const vector<string>& record,set<string>& result,const vector<string>& field_vec)
{
result.clear();
if(record.size()!=field_vec.size())
return -1;
for(int i=0;i<field_vec.size()-1;i++) // last field is label
{
if(field_vec[i].substr(0,3)==string("tag"))
{
if(record[i]=="1")
result.insert(field_vec[i]);
}else{
result.insert(field_vec[i]+":"+record[i]);
```

```
}
}
return 0;
}
/*
int LogisticRegression::read_dataset(const char* filename, const char* configure)
{
map<string,string> conf_dict;
int status=CommonTool::load_configure(configure,conf_dict);
if(status<0)
{
cout<<"load configure file:"<<configure<<" faild."<<endl;
return -1;
}
vector<string> field_vec;
int n=CommonTool::split(conf_dict["filter:field_name_vec"],',',field_vec);
string line;
ifstream fin(filename);
int cnt=0;
while(getline(fin,line))
{
cnt++;
vector<string> record;
set<string> result;
CommonTool::split(line,'\t',record);
this->labels.push_back(atoi(record.back().c_str()));
if(record.size()!=field_vec.size())
{
cout<<"error line:(fields="<<record.size()<<",required
"<<field_vec.size()<<")"<<line<<endl;
continue;
}
convert_record(record,result,field_vec);
```

```
this->dataset.push_back(result);
if(cnt%1000000==0)
{
time_t t=time(NULL);
cout<<"progress:"<<cnt<<" time now:"<<asctime(localtime(&t))<<endl;
}
}
cout<<"read records:"<<cnt<<endl;
}
*/
```

EK-E

E.1: Gradyan Arttırılmış Ağaçlar Python Kodu

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('data.csv')
X = dataset.iloc[:, 3:13].values
y = dataset.iloc[:, 13].values

# Encoding categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X_1 = LabelEncoder()

X[:, 1] = labelencoder_X_1.fit_transform(X[:, 1])
labelencoder_X_2 = LabelEncoder()

X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])
onehotencoder = OneHotEncoder(categorical_features = [1])

X = onehotencoder.fit_transform(X).toarray()
X = X[:, 1:]

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size = 0.2, random_state = 0)

# Fitting XGBoost to the training data
import xgboost as xgb
my_model = xgb.XGBClassifier()
```

```
my_model.fit(X_train, y_train)

# Predicting the Test set results
y_pred = my_model.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

EK-F

F.1: Destek Vektör Makinesi Python Kodu

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd



datasets = pd.read_csv('dataset.csv')
X = datasets.iloc[:, [2,3]].values
Y = datasets.iloc[:, 4].values

from sklearn.model_selection import train_test_split
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, Y, test_size = 0.25, random_state
    = 0)

from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_Train = sc_X.fit_transform(X_Train)
X_Test = sc_X.transform(X_Test)

from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_Train, Y_Train)

Y_Pred = classifier.predict(X_Test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_Test, Y_Pred)

from matplotlib.colors import ListedColormap
X_Set, Y_Set = X_Train, Y_Train
X1, X2 = np.meshgrid(np.arange(start = X_Set[:, 0].min() - 1, stop = X_Set[:, 0].max() +
    1, step = 0.01),
                     np.arange(start = X_Set[:, 1].min() - 1, stop = X_Set[:, 1].max() + 1, step =
    0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
    X2.ravel()]).T).reshape(X1.shape),
```

```
        alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(Y_Set)):
    plt.scatter(X_Set[Y_Set == j, 0], X_Set[Y_Set == j, 1],
            c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Support Vector Machine (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

from matplotlib.colors import ListedColormap
X_Set, Y_Set = X_Test, Y_Test
X1, X2 = np.meshgrid(np.arange(start = X_Set[:, 0].min() - 1, stop = X_Set[:, 0].max() +
    1, step = 0.01),
                np.arange(start = X_Set[:, 1].min() - 1, stop = X_Set[:, 1].max() + 1, step =
    0.01))
plt.contourf(X1,                X2,                classifier.predict(np.array([X1.ravel(),
    X2.ravel()]).T).reshape(X1.shape),
        alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(Y_Set)):
    plt.scatter(X_Set[Y_Set == j, 0], X_Set[Y_Set == j, 1],
            c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Support Vector Machine (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

EK-G1

G1: Konvolüsyonel Sinir Ağları Sınıflandırıcısı

```
import torch

from . import networks

from os.path import join

from util.util import seg_accuracy, print_network


class ClassifierModel:

""" Class for training Model weights

:args opt: structure containing configuration params

e.g.,
```

```
--dataset_mode -> classification / segmentation)
--arch -> network type
"""
def __init__(self, opt):
self.opt = opt
self.gpu_ids = opt.gpu_ids
self.is_train = opt.is_train
self.device = torch.device('cuda:{}'.format(self.gpu_ids[0])) if self.gpu_ids else torch.device('cpu')
self.save_dir = join(opt.checkpoints_dir, opt.name)
self.optimizer = None
self.edge_features = None
self.labels = None
self.mesh = None
self.soft_label = None
self.loss = None

#
self.nclasses = opt.nclasses

# load/define networks
self.net = networks.define_classifier(opt.input_nc, opt.ncf, opt.ninput_edges, opt.nclasses, opt,
self.gpu_ids, opt.arch, opt.init_type, opt.init_gain)
self.net.train(self.is_train)
self.criterion = networks.define_loss(opt).to(self.device)

if self.is_train:
self.optimizer = torch.optim.Adam(self.net.parameters(), lr=opt.lr, betas=(opt.beta1, 0.999))
self.scheduler = networks.get_scheduler(self.optimizer, opt)
print_network(self.net)

if not self.is_train or opt.continue_train:
self.load_network(opt.which_epoch)

def set_input(self, data):
```

```
input_edge_features = torch.from_numpy(data['edge_features']).float()
labels = torch.from_numpy(data['label']).long()
# set inputs
self.edge_features = input_edge_features.to(self.device).requires_grad_(self.is_train)
self.labels = labels.to(self.device)
self.mesh = data['mesh']
if self.opt.dataset_mode == 'segmentation' and not self.is_train:
self.soft_label = torch.from_numpy(data['soft_label'])

def forward(self):
out = self.net(self.edge_features, self.mesh)
return out

def backward(self, out):
self.loss = self.criterion(out, self.labels)
self.loss.backward()

def optimize_parameters(self):
self.optimizer.zero_grad()
out = self.forward()
self.backward(out)
self.optimizer.step()

##################

def load_network(self, which_epoch):
"""load model from disk"""
save_filename = '%s_net.pth' % which_epoch
load_path = join(self.save_dir, save_filename)
net = self.net
if isinstance(net, torch.nn.DataParallel):
net = net.module
print('loading the model from %s' % load_path)
# PyTorch newer than 0.4 (e.g., built from
# GitHub source), you can remove str() on self.device
```

```
state_dict = torch.load(load_path, map_location=str(self.device))
if hasattr(state_dict, '_metadata'):
del state_dict._metadata
net.load_state_dict(state_dict)

def save_network(self, which_epoch):
"""save model to disk"""
save_filename = '%s_net.pth' % (which_epoch)
save_path = join(self.save_dir, save_filename)
if len(self.gpu_ids) > 0 and torch.cuda.is_available():
torch.save(self.net.module.cpu().state_dict(), save_path)
self.net.cuda(self.gpu_ids[0])
else:
torch.save(self.net.cpu().state_dict(), save_path)

def update_learning_rate(self):
"""update learning rate (called once every epoch)"""
self.scheduler.step()
lr = self.optimizer.param_groups[0]['lr']
print('learning rate = %.7f' % lr)

def test(self):
"""tests model
returns: number correct and total number
"""
with torch.no_grad():
out = self.forward()
# compute number of correct
pred_class = out.data.max(1)[1]
label_class = self.labels
self.export_segmentation(pred_class.cpu())
correct = self.get_accuracy(pred_class, label_class)
return correct, len(label_class)

def get_accuracy(self, pred, labels):
```

```
    """computes accuracy for classification / segmentation """
    if self.opt.dataset_mode == 'classification':
    correct = pred.eq(labels).sum()
    elif self.opt.dataset_mode == 'segmentation':
    correct = seg_accuracy(pred, self.soft_label, self.mesh)
    return correct

    def export_segmentation(self, pred_seg):
    if self.opt.dataset_mode == 'segmentation':
    for meshi, mesh in enumerate(self.mesh):
mesh.export_segments(pred_seg[meshi, :])
```

EK-G2

G2: Konvolüsyonel Sinir Ağı

```
    import torch
    import torch.nn as nn
    from torch.nn import init
    import functools
    from torch.optim import lr_scheduler
    from models.layers.mesh_conv import MeshConv
    import torch.nn.functional as F
    from models.layers.mesh_pool import MeshPool
    from models.layers.mesh_unpool import MeshUnpool

    ###############################################################################
    # Helper Functions
    ###############################################################################

    def get_norm_layer(norm_type='instance', num_groups=1):
    if norm_type == 'batch':
    norm_layer = functools.partial(nn.BatchNorm2d, affine=True)
    elif norm_type == 'instance':
    norm_layer = functools.partial(nn.InstanceNorm2d, affine=False)
```

```
elif norm_type == 'group':
norm_layer = functools.partial(nn.GroupNorm, affine=True,
num_groups=num_groups)
elif norm_type == 'none':
norm_layer = NoNorm
else:
raise NotImplementedError('normalization layer [%s] is not found' % norm_type)
return norm_layer

def get_norm_args(norm_layer, nfeats_list):
if hasattr(norm_layer, '__name__') and norm_layer.__name__ == 'NoNorm':
norm_args = [{'fake': True} for f in nfeats_list]
elif norm_layer.func.__name__ == 'GroupNorm':
norm_args = [{'num_channels': f} for f in nfeats_list]
elif norm_layer.func.__name__ == 'BatchNorm':
norm_args = [{'num_features': f} for f in nfeats_list]
else:
raise NotImplementedError('normalization layer [%s] is not found' %
norm_layer.func.__name__)
return norm_args

class NoNorm(nn.Module): #todo with abstractclass and pass
def __init__(self, fake=True):
self.fake = fake
super(NoNorm, self).__init__()
def forward(self, x):
return x
def __call__(self, x):
return self.forward(x)

def get_scheduler(optimizer, opt):
if opt.lr_policy == 'lambda':
def lambda_rule(epoch):
lr_l = 1.0 - max(0, epoch + 1 + opt.epoch_count - opt.niter) / float(opt.niter_decay +
1)
```

return lr_l

scheduler = lr_scheduler.LambdaLR(optimizer, lr_lambda=lambda_rule)

elif opt.lr_policy == 'step':

scheduler = lr_scheduler.StepLR(optimizer, step_size=opt.lr_decay_iters, gamma=0.1)

elif opt.lr_policy == 'plateau':

scheduler = lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.2, threshold=0.01, patience=5)

else:

return NotImplementedError('learning rate policy [%s] is not implemented', opt.lr_policy)

return scheduler

def init_weights(net, init_type, init_gain):

def init_func(m):

classname = m.__class__.__name__

if hasattr(m, 'weight') and (classname.find('Conv') != -1 or classname.find('Linear') != -1):

if init_type == 'normal':

init.normal_(m.weight.data, 0.0, init_gain)

elif init_type == 'xavier':

init.xavier_normal_(m.weight.data, gain=init_gain)

elif init_type == 'kaiming':

init.kaiming_normal_(m.weight.data, a=0, mode='fan_in')

elif init_type == 'orthogonal':

init.orthogonal_(m.weight.data, gain=init_gain)

else:

raise NotImplementedError('initialization method [%s] is not implemented' % init_type)

elif classname.find('BatchNorm2d') != -1:

init.normal_(m.weight.data, 1.0, init_gain)

init.constant_(m.bias.data, 0.0)

net.apply(init_func)

def init_net(net, init_type, init_gain, gpu_ids):

```
if len(gpu_ids) > 0:
assert(torch.cuda.is_available())
net.cuda(gpu_ids[0])
net = net.cuda()
net = torch.nn.DataParallel(net, gpu_ids)
if init_type != 'none':
init_weights(net, init_type, init_gain)
return net

def define_classifier(input_nc, ncf, ninput_edges, nclasses, opt, gpu_ids, arch,
init_type, init_gain):
net = None
norm_layer = get_norm_layer(norm_type=opt.norm, num_groups=opt.num_groups)

if arch == 'mconvnet':
net = MeshConvNet(norm_layer, input_nc, ncf, nclasses, ninput_edges, opt.pool_res,
opt.fc_n,
opt.resblocks)
elif arch == 'meshunet':
down_convs = [input_nc] + ncf
up_convs = ncf[::-1] + [nclasses]
pool_res = [ninput_edges] + opt.pool_res
net = MeshEncoderDecoder(pool_res, down_convs, up_convs, blocks=opt.resblocks,
transfer_data=True)
else:
raise NotImplementedError('Encoder model name [%s] is not recognized' % arch)
return init_net(net, init_type, init_gain, gpu_ids)

def define_loss(opt):
if opt.dataset_mode == 'classification':
loss = torch.nn.CrossEntropyLoss()
elif opt.dataset_mode == 'segmentation':
loss = torch.nn.CrossEntropyLoss(ignore_index=-1)
return loss
```

```
##############################################################################
##########
# Classes For Classification / Segmentation Networks
##############################################################################
##########

class MeshConvNet(nn.Module):
"""Network for learning a global shape descriptor (classification)
"""
def __init__(self, norm_layer, nf0, conv_res, nclasses, input_res, pool_res, fc_n,
nresblocks=3):
super(MeshConvNet, self).__init__()
self.k = [nf0] + conv_res
self.res = [input_res] + pool_res
norm_args = get_norm_args(norm_layer, self.k[1:])

for i, ki in enumerate(self.k[:-1]):
setattr(self, 'conv{}'.format(i), MResConv(ki, self.k[i + 1], nresblocks))
setattr(self, 'norm{}'.format(i), norm_layer(**norm_args[i]))
setattr(self, 'pool{}'.format(i), MeshPool(self.res[i + 1]))


self.gp = torch.nn.AvgPool1d(self.res[-1])
# self.gp = torch.nn.MaxPool1d(self.res[-1])
self.fc1 = nn.Linear(self.k[-1], fc_n)
self.fc2 = nn.Linear(fc_n, nclasses)

def forward(self, x, mesh):

for i in range(len(self.k) - 1):
x = getattr(self, 'conv{}'.format(i))(x, mesh)
x = F.relu(getattr(self, 'norm{}'.format(i))(x))
x = getattr(self, 'pool{}'.format(i))(x, mesh)

x = self.gp(x)
x = x.view(-1, self.k[-1])

x = F.relu(self.fc1(x))
```

```
x = self.fc2(x)
return x

class MResConv(nn.Module):
def __init__(self, in_channels, out_channels, skips=1):
super(MResConv, self).__init__()
self.in_channels = in_channels
self.out_channels = out_channels
self.skips = skips
self.conv0 = MeshConv(self.in_channels, self.out_channels, bias=False)
for i in range(self.skips):
setattr(self, 'bn{}'.format(i + 1), nn.BatchNorm2d(self.out_channels))
setattr(self, 'conv{}'.format(i + 1),
MeshConv(self.out_channels, self.out_channels, bias=False))

def forward(self, x, mesh):
x = self.conv0(x, mesh)
x1 = x
for i in range(self.skips):
x = getattr(self, 'bn{}'.format(i + 1))(F.relu(x))
x = getattr(self, 'conv{}'.format(i + 1))(x, mesh)
x += x1
x = F.relu(x)
return x

class MeshEncoderDecoder(nn.Module):
"""Network for fully-convolutional tasks (segmentation)
"""
def __init__(self, pools, down_convs, up_convs, blocks=0, transfer_data=True):
super(MeshEncoderDecoder, self).__init__()
self.transfer_data = transfer_data
self.encoder = MeshEncoder(pools, down_convs, blocks=blocks)
unrolls = pools[:-1].copy()
unrolls.reverse()
```

```
self.decoder = MeshDecoder(unrolls, up_convs, blocks=blocks,
transfer_data=transfer_data)

def forward(self, x, meshes):
fe, before_pool = self.encoder((x, meshes))
fe = self.decoder((fe, meshes), before_pool)
return fe

def __call__(self, x, meshes):
return self.forward(x, meshes)

class DownConv(nn.Module):
def __init__(self, in_channels, out_channels, blocks=0, pool=0):
super(DownConv, self).__init__()
self.bn = []
self.pool = None
self.conv1 = MeshConv(in_channels, out_channels)
self.conv2 = []
for _ in range(blocks):
self.conv2.append(MeshConv(out_channels, out_channels))
self.conv2 = nn.ModuleList(self.conv2)
for _ in range(blocks + 1):
self.bn.append(nn.InstanceNorm2d(out_channels))
self.bn = nn.ModuleList(self.bn)
if pool:
self.pool = MeshPool(pool)

def __call__(self, x):
return self.forward(x)

def forward(self, x):
fe, meshes = x
x1 = self.conv1(fe, meshes)
if self.bn:
x1 = self.bn[0](x1)
x1 = F.relu(x1)
```

```
x2 = x1
for idx, conv in enumerate(self.conv2):
x2 = conv(x1, meshes)
if self.bn:
x2 = self.bn[idx + 1](x2)
x2 = x2 + x1
x2 = F.relu(x2)
x1 = x2
x2 = x2.squeeze(3)
before_pool = None
if self.pool:
before_pool = x2
x2 = self.pool(x2, meshes)
return x2, before_pool

class UpConv(nn.Module):
def __init__(self, in_channels, out_channels, blocks=0, unroll=0, residual=True,
batch_norm=True, transfer_data=True):
super(UpConv, self).__init__()
self.residual = residual
self.bn = []
self.unroll = None
self.transfer_data = transfer_data
self.up_conv = MeshConv(in_channels, out_channels)
if transfer_data:
self.conv1 = MeshConv(2 * out_channels, out_channels)
else:
self.conv1 = MeshConv(out_channels, out_channels)
self.conv2 = []
for _ in range(blocks):
self.conv2.append(MeshConv(out_channels, out_channels))
self.conv2 = nn.ModuleList(self.conv2)
if batch_norm:
```

```
for _ in range(blocks + 1):
self.bn.append(nn.InstanceNorm2d(out_channels))
self.bn = nn.ModuleList(self.bn)
if unroll:
self.unroll = MeshUnpool(unroll)

def __call__(self, x, from_down=None):
return self.forward(x, from_down)

def forward(self, x, from_down):
from_up, meshes = x
x1 = self.up_conv(from_up, meshes).squeeze(3)
if self.unroll:
x1 = self.unroll(x1, meshes)
if self.transfer_data:
x1 = torch.cat((x1, from_down), 1)
x1 = self.conv1(x1, meshes)
if self.bn:
x1 = self.bn[0](x1)
x1 = F.relu(x1)
x2 = x1
for idx, conv in enumerate(self.conv2):
x2 = conv(x1, meshes)
if self.bn:
x2 = self.bn[idx + 1](x2)
if self.residual:
x2 = x2 + x1
x2 = F.relu(x2)
x1 = x2
x2 = x2.squeeze(3)
return x2

class MeshEncoder(nn.Module):
def __init__(self, pools, convs, fcs=None, blocks=0, global_pool=None):
```

```
super(MeshEncoder, self).__init__()
self.fcs = None
self.convs = []
for i in range(len(convs) - 1):
if i + 1 < len(pools):
pool = pools[i + 1]
else:
pool = 0
self.convs.append(DownConv(convs[i], convs[i + 1], blocks=blocks, pool=pool))
self.global_pool = None
if fcs is not None:
self.fcs = []
self.fcs_bn = []
last_length = convs[-1]
if global_pool is not None:
if global_pool == 'max':
self.global_pool = nn.MaxPool1d(pools[-1])
elif global_pool == 'avg':
self.global_pool = nn.AvgPool1d(pools[-1])
else:
assert False, 'global_pool %s is not defined' % global_pool
else:
last_length *= pools[-1]
if fcs[0] == last_length:
fcs = fcs[1:]
for length in fcs:
self.fcs.append(nn.Linear(last_length, length))
self.fcs_bn.append(nn.InstanceNorm1d(length))
last_length = length
self.fcs = nn.ModuleList(self.fcs)
self.fcs_bn = nn.ModuleList(self.fcs_bn)
self.convs = nn.ModuleList(self.convs)
reset_params(self)
```

```
def forward(self, x):
fe, meshes = x
encoder_outs = []
for conv in self.convs:
fe, before_pool = conv((fe, meshes))
encoder_outs.append(before_pool)
if self.fcs is not None:
if self.global_pool is not None:
fe = self.global_pool(fe)
fe = fe.contiguous().view(fe.size()[0], -1)
for i in range(len(self.fcs)):
fe = self.fcs[i](fe)
if self.fcs_bn:
x = fe.unsqueeze(1)
fe = self.fcs_bn[i](x).squeeze(1)
if i < len(self.fcs) - 1:
fe = F.relu(fe)
return fe, encoder_outs

def __call__(self, x):
return self.forward(x)

class MeshDecoder(nn.Module):
def __init__(self, unrolls, convs, blocks=0, batch_norm=True, transfer_data=True):
super(MeshDecoder, self).__init__()
self.up_convs = []
for i in range(len(convs) - 2):
if i < len(unrolls):
unroll = unrolls[i]
else:
unroll = 0
self.up_convs.append(UpConv(convs[i], convs[i + 1], blocks=blocks, unroll=unroll,
batch_norm=batch_norm, transfer_data=transfer_data))
```

```
        self.final_conv = UpConv(convs[-2], convs[-1], blocks=blocks, unroll=False,
        batch_norm=batch_norm, transfer_data=False)
        self.up_convs = nn.ModuleList(self.up_convs)
        reset_params(self)

        def forward(self, x, encoder_outs=None):
        fe, meshes = x
        for i, up_conv in enumerate(self.up_convs):
        before_pool = None
        if encoder_outs is not None:
        before_pool = encoder_outs[-(i+2)]
        fe = up_conv((fe, meshes), before_pool)
        fe = self.final_conv((fe, meshes))
        return fe

        def __call__(self, x, encoder_outs=None):
        return self.forward(x, encoder_outs)

        def reset_params(model): # todo replace with my init
        for i, m in enumerate(model.modules()):
        weight_init(m)

        def weight_init(m):
        if isinstance(m, nn.Conv2d):
        nn.init.xavier_normal_(m.weight)
nn.init.constant_(m.bias, 0)
```

EK-H1

H1: Bin Median Fonksiyonu

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn import linear_model
# import statsmodels.api as sm
import statistics
import math
from collections import OrderedDict


x =[]
print("enter the data")
x = list(map(float, input().split()))

print("enter the number of bins")
bi = int(input())

# X_dict will store the data in sorted order
X_dict = OrderedDict()
# x_old will store the original data
x_old ={}
# x_new will store the data after binning
x_new ={}

for i in range(len(x)):
	X_dict[i]= x[i]
	x_old[i]= x[i]

x_dict = sorted(X_dict.items(), key = lambda x: x[1])


# list of lists(bins)
binn =[]
# a variable to find the mean of each bin
avrg =[]

i = 0
k = 0
num_of_data_in_each_bin = int(math.ceil(len(x)/bi))
# performing binning
for g, h in X_dict.items():
	if(i<num_of_data_in_each_bin):
		avrg.append(h)
		i = i + 1
	elif(i == num_of_data_in_each_bin):
		k = k + 1
		i = 0
```

```
            binn.append(statistics.median(avrg))
            avrg =[]
            avrg.append(h)
            i = i + 1

binn.append(statistics.median(avrg))

# store the new value of each data
i = 0
j = 0
for g, h in X_dict.items():
    if(i<num_of_data_in_each_bin):
            x_new[g]= round(binn[j], 3)
            i = i + 1
    else:
            i = 0
            j = j + 1
            x_new[g]= round(binn[j], 3)
            i = i + 1

print("number of data in each bin")
print(math.ceil(len(x)/bi))
for i in range(0, len(x)):
    print('index {2} old value {0} new value {1}'.format(x_old[i], x_new[i], i))
```

EK-H2

H2: Bin Boundary Fonksiyonu

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn import linear_model
# import statsmodels.api as sm
import statistics
import math
from collections import OrderedDict

x =[]
print("enter the data")
x = list(map(float, input().split()))

print("enter the number of bins")
bi = int(input())

# X_dict will store the data in sorted order
X_dict = OrderedDict()
# x_old will store the original data
x_old ={}
# x_new will store the data after binning
x_new ={}
```

```
for i in range(len(x)):
	X_dict[i]= x[i]
	x_old[i]= x[i]

x_dict = sorted(X_dict.items(), key = lambda x: x[1])

# list of lists(bins)
binn =[]
# a variable to find the mean of each bin
avrg =[]

i = 0
k = 0
num_of_data_in_each_bin = int(math.ceil(len(x)/bi))

for g, h in X_dict.items():
	if(i<num_of_data_in_each_bin):
		avrg.append(h)
		i = i + 1
	elif(i == num_of_data_in_each_bin):
		k = k + 1
		i = 0
		binn.append([min(avrg), max(avrg)])
		avrg =[]
		avrg.append(h)
		i = i + 1
binn.append([min(avrg), max(avrg)])

i = 0
j = 0

for g, h in X_dict.items():
	if(i<num_of_data_in_each_bin):
		if(abs(h-binn[j][0]) >= abs(h-binn[j][1])):
			x_new[g]= binn[j][1]
			i = i + 1
		else:
			x_new[g]= binn[j][0]
			i = i + 1
	else:
		i = 0
		j = j + 1
		if(abs(h-binn[j][0]) >= abs(h-binn[j][1])):
			x_new[g]= binn[j][1]
		else:
			x_new[g]= binn[j][0]
		i = i + 1

print("number of data in each bin")
```

```
print(math.ceil(len(x)/bi))
for i in range(0, len(x)):
    print('index {2} old value {0} new value {1}'.format(x_old[i], x_new[i], i))
```

EK-H3

H3: Bin Mean(Ortalama) Fonksiyonu

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn import linear_model
# import statsmodels.api as sm
import statistics
import math
from collections import OrderedDict

x =[]
print("enter the data")
x = list(map(float, input().split()))

print("enter the number of bins")
bi = int(input())

# X_dict will store the data in sorted order
X_dict = OrderedDict()
# x_old will store the original data
x_old ={}
# x_new will store the data after binning
x_new ={}


for i in range(len(x)):
    X_dict[i]= x[i]
    x_old[i]= x[i]

x_dict = sorted(X_dict.items(), key = lambda x: x[1])

# list of lists(bins)
binn =[]
# a variable to find the mean of each bin
avrg = 0

i = 0
k = 0
num_of_data_in_each_bin = int(math.ceil(len(x)/bi))

# performing binning
for g, h in X_dict.items():
    if(i<num_of_data_in_each_bin):
```

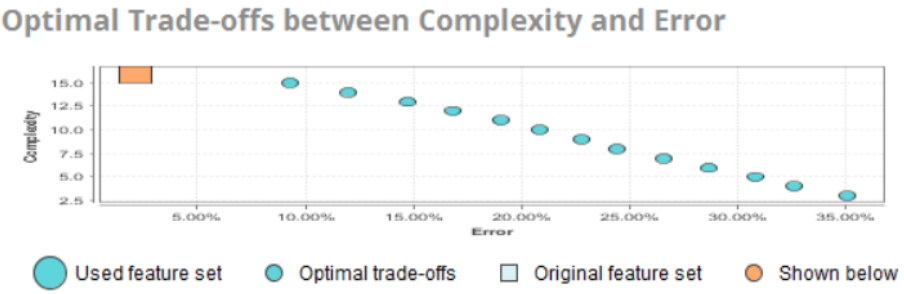
```
            avrg = avrg + h
            i = i + 1
    elif(i == num_of_data_in_each_bin):
            k = k + 1
            i = 0
            binn.append(round(avrg / num_of_data_in_each_bin, 3))
            avrg = 0
            avrg = avrg + h
            i = i + 1
rem = len(x)% bi
if(rem == 0):
    binn.append(round(avrg / num_of_data_in_each_bin, 3))
else:
    binn.append(round(avrg / rem, 3))

# store the new value of each data
i = 0
j = 0
for g, h in X_dict.items():
    if(i<num_of_data_in_each_bin):
            x_new[g]= binn[j]
            i = i + 1
    else:
            i = 0
            j = j + 1
            x_new[g]= binn[j]
            i = i + 1
print("number of data in each bin")
print(math.ceil(len(x)/bi))

for i in range(0, len(x)):
    print('index {2} old value {0} new value {1}'.format(x_old[i], x_new[i], i))
```

EK I – Ekstradan Bulgular

I.1: Naif Bayes

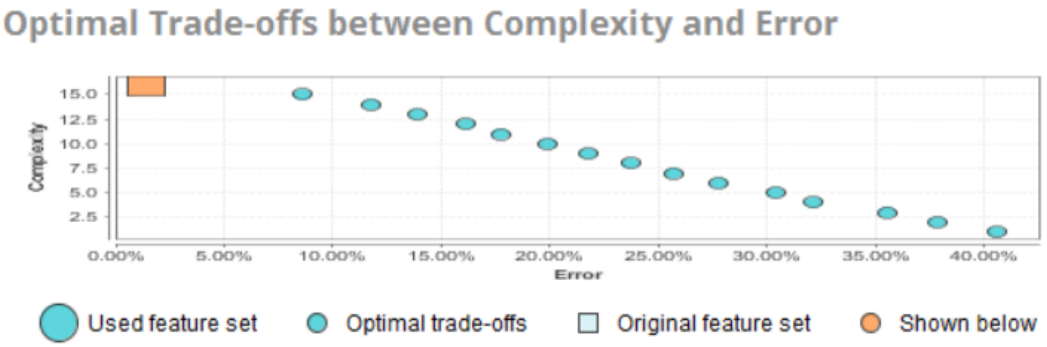


Bu grafik, özellik mühendisliği çalışmasının sonucunu gösterir. Her nokta farklı bir özellik kümesini, yani orijinal sütunların bir alt kümesini veya hatta özellik oluşturma durumunda yeni oluşturulan bazı özellikleri içeren bir grubu temsil eder. Bir özellik seti, örneğin 5'in karmaşıklığına sahip olabilir ve % 18 ya da 0.18 hata

oranına ulaşabilir. Bu hata oranının, özellik mühendisliğinin kendisinin eğitim hata oranı olduğunu unutmamak gerekir. Bir uzatma setinde doğrulanan test hatası, bundan daha yüksek olacaktır ve performans sekmesinde ve modele genel bakışta gösterilecektir. En iyi özellik setleri, düşük karmaşıklık ve düşük hata oranlarıyla sol alt köşededir. Aynı anda hata oranını arttırmadan karmaşıklığı en aza indirmenin mümkün olmadığını unutmamak gerekir. Bununla birlikte, birçok durumda, ortaya çıkan özellik boşlukları (yuvarlak noktalar) daha az karmaşıktır ve yine de orijinal özellik alanından (kare) daha hassastır. Daha az özellik kullanmak, modellerin daha hızlı eğitilebileceği anlamına gelir. Son modeli oluşturmak için kullanılmış olan özellik seti daha büyük gösterilmiştir.

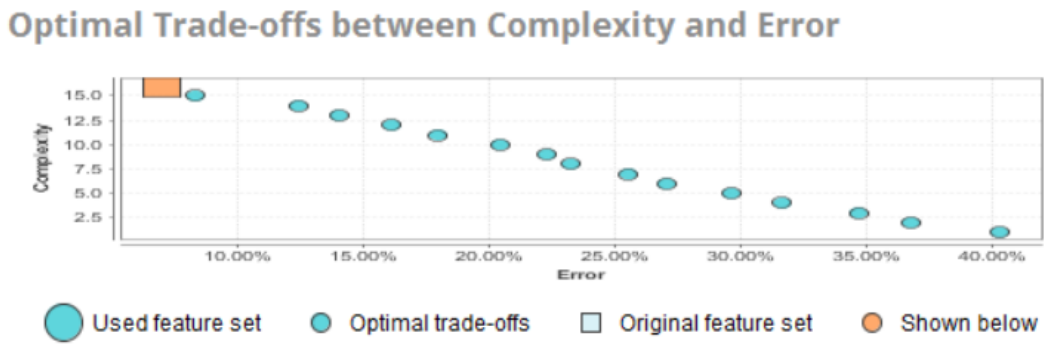
Name	Expression	Complexity
0.215261493	[0.215261493]	1
0.329169068	[0.329169068]	1
0.788656772	[0.788656772]	1
0.950809496	[0.950809496]	1
0.591770924	[0.591770924]	1
0.086314274	[0.086314274]	1
0.804403656	[0.804403656]	1
0.350843754	[0.350843754]	1
0.09923022	[0.09923022]	1
0.04058144	[0.04058144]	1
0.218576167	[0.218576167]	1
0.090783852	[0.090783852]	1
0.937680009	[0.937680009]	1
0.495146927	[0.495146927]	1
0.340776522	[0.340776522]	1
0.304858309	[0.304858309]	1

I2: Genelleştirilmiş Doğrusal Model



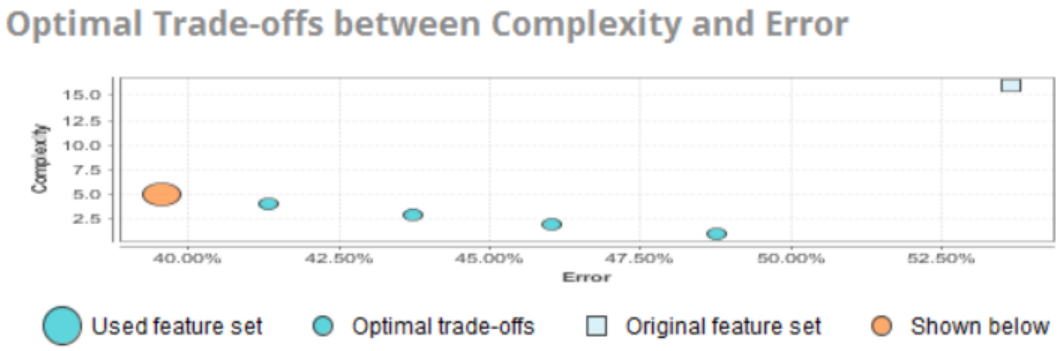
Name	Expression	Complexity
0.215261493	[0.215261493]	1
0.329169068	[0.329169068]	1
0.788656772	[0.788656772]	1
0.950809496	[0.950809496]	1
0.591770924	[0.591770924]	1
0.086314274	[0.086314274]	1
0.804403656	[0.804403656]	1
0.350843754	[0.350843754]	1
0.09923022	[0.09923022]	1
0.04058144	[0.04058144]	1
0.218576167	[0.218576167]	1
0.090783852	[0.090783852]	1
0.937680009	[0.937680009]	1
0.495146927	[0.495146927]	1
0.340776522	[0.340776522]	1
0.304858309	[0.304858309]	1

I3:Lojistik Regresyon



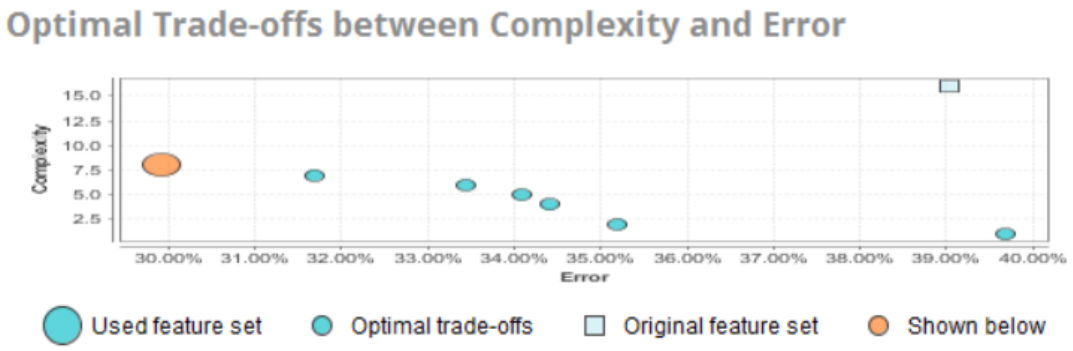
Name	Expression	Complexity
0.215261493	[0.215261493]	1
0.329169068	[0.329169068]	1
0.788656772	[0.788656772]	1
0.950809496	[0.950809496]	1
0.591770924	[0.591770924]	1
0.086314274	[0.086314274]	1
0.804403656	[0.804403656]	1
0.350843754	[0.350843754]	1
0.09923022	[0.09923022]	1
0.04058144	[0.04058144]	1
0.218576167	[0.218576167]	1
0.090783852	[0.090783852]	1
0.937680009	[0.937680009]	1
0.495146927	[0.495146927]	1
0.340776522	[0.340776522]	1
0.304858309	[0.304858309]	1

I4:Karar Ağaçları



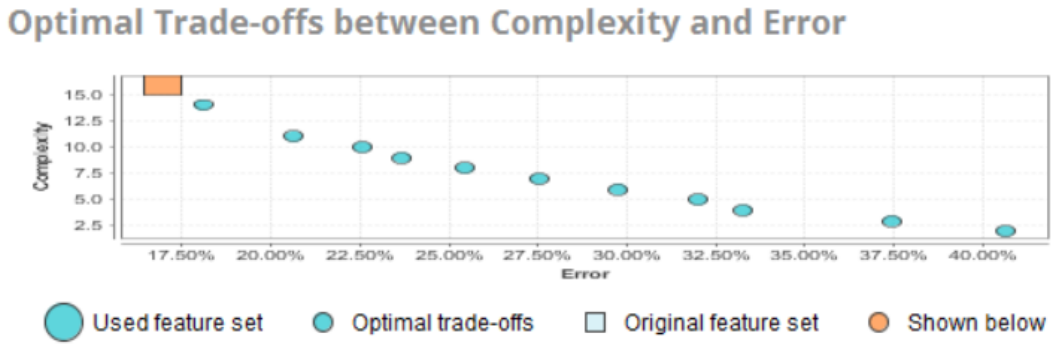
Name	Expression	Complexity
0.937680009	[0.937680009]	1
0.495146927	[0.495146927]	1
0.09923022	[0.09923022]	1
0.218576167	[0.218576167]	1
0.350843754	[0.350843754]	1

I5:Random Forest



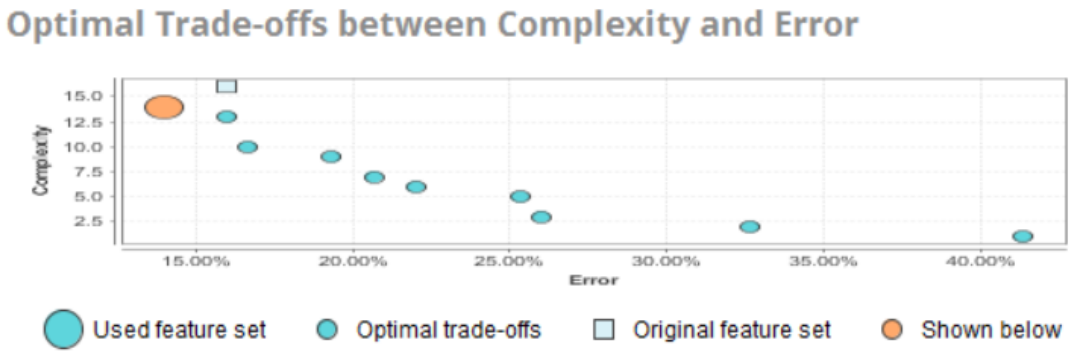
Name	Expression	Complexity
0.329169068	[0.329169068]	1
0.09923022	[0.09923022]	1
GenSym111	[0.04058144]*[0.215261493]	2
0.090783852	[0.090783852]	1
0.086314274	[0.086314274]	1
0.804403656	[0.804403656]	1
0.350843754	[0.350843754]	1

I6: Gradyan Arttırılmış Ağaçlar



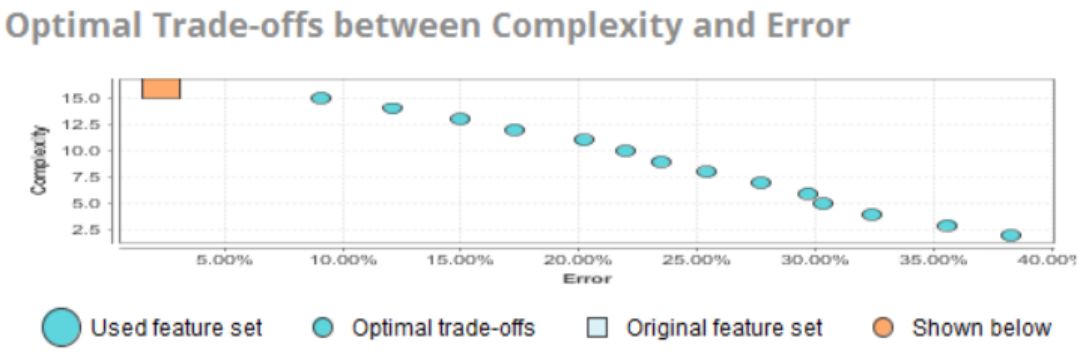
Name	Expression	Complexity
0.215261493	[0.215261493]	1
0.329169068	[0.329169068]	1
0.788656772	[0.788656772]	1
0.950809496	[0.950809496]	1
0.591770924	[0.591770924]	1
0.086314274	[0.086314274]	1
0.804403656	[0.804403656]	1
0.350843754	[0.350843754]	1
0.09923022	[0.09923022]	1
0.04058144	[0.04058144]	1
0.218576167	[0.218576167]	1
0.090783852	[0.090783852]	1
0.937680009	[0.937680009]	1
0.495146927	[0.495146927]	1
0.340776522	[0.340776522]	1
0.304858309	[0.304858309]	1

I7: Destek Vektör Makinesi



Name	Expression	Complexity
0.215261493	[0.215261493]	1
0.788656772	[0.788656772]	1
0.950809496	[0.950809496]	1
0.591770924	[0.591770924]	1
0.804403656	[0.804403656]	1
0.350843754	[0.350843754]	1
0.09923022	[0.09923022]	1
0.04058144	[0.04058144]	1
0.218576167	[0.218576167]	1
0.090783852	[0.090783852]	1
0.937680009	[0.937680009]	1
0.495146927	[0.495146927]	1
0.340776522	[0.340776522]	1
0.304858309	[0.304858309]	1

I8: Hızlı Büyük Marj



Name	Expression	Complexity
0.215261493	[0.215261493]	1
0.329169068	[0.329169068]	1
0.788656772	[0.788656772]	1
0.950809496	[0.950809496]	1
0.591770924	[0.591770924]	1
0.086314274	[0.086314274]	1
0.804403656	[0.804403656]	1
0.350843754	[0.350843754]	1
0.09923022	[0.09923022]	1
0.04058144	[0.04058144]	1
0.218576167	[0.218576167]	1
0.090783852	[0.090783852]	1
0.937680009	[0.937680009]	1
0.495146927	[0.495146927]	1
0.340776522	[0.340776522]	1
0.304858309	[0.304858309]	1

I9: Konvolüsyonel Sinir Ağı

Name	Expression	Complexity
0.215261493	[0.215261493]	1
0.329169068	[0.329169068]	1
0.788656772	[0.788656772]	1
0.950809496	[0.950809496]	1
0.591770924	[0.591770924]	1
0.086314274	[0.086314274]	1
0.804403656	[0.804403656]	1
0.350843754	[0.350843754]	1
0.09923022	[0.09923022]	1
0.04058144	[0.04058144]	1
0.218576167	[0.218576167]	1
0.090783852	[0.090783852]	1
0.937680009	[0.937680009]	1
0.495146927	[0.495146927]	1
0.340776522	[0.340776522]	1
0.304858309	[0.304858309]	1

Model Metrics Type: Binomial
Description: Metrics reported on temporary training frame with 10040 samples
model id: rm-h2o-model-model-390517
frame id: rm-h2o-frame-model-615009.temporary.sample.28.49%
MSE: 0.0026381898
R^2: 0.98944676
AUC: 0.99996436
logloss: 0.009106332
CM: Confusion Matrix (vertical: actual; across: predicted):

	range2	range1	Error	Rate
range2	4964	22	0.0044	= 22 / 4,986
range1	6	5048	0.0012	= 6 / 5,054
Totals	4970	5070	0.0028	= 28 / 10,040

Gains/Lift Table (Avg response rate: 50.34 %):

Group	Cumulative Data Fraction	Lower Threshold	Lift	Cumulative Lift	Response Rate	Cumulative Response Rate	Capture Rate	Cumulative Capture Rate	Gain	Cumulative Gain
1	0.31424303	1.000000	1.986545	1.986545	1.000000	1.000000	0.624258	0.624258	98.654531	98.654531
2	0.40000000	1.000000	1.986545	1.986545	1.000000	1.000000	0.170360	0.794618	98.654531	98.654531
3	0.50000000	0.612599	1.974674	1.984171	0.994024	0.998805	0.197467	0.992085	97.467353	98.417095
4	0.60000000	0.000000	0.079145	1.666667	0.039841	0.838977	0.007915	1.000000	-92.085477	66.666667
5	0.70000000	0.000000	0.000000	1.428571	0.000000	0.719124	0.000000	1.000000	-100.000000	42.857143
6	0.80000000	0.000000	0.000000	1.250000	0.000000	0.629233	0.000000	1.000000	-100.000000	25.000000
7	0.90000000	0.000000	0.000000	1.111111	0.000000	0.559318	0.000000	1.000000	-100.000000	11.111111
8	1.00000000	0.000000	0.000000	1.000000	0.000000	0.503386	0.000000	1.000000	-100.000000	0.000000

Status of Neuron Layers (predicting 0, 2-class classification, bernoulli distribution, CrossEntropy loss, 3,502 weights/biases, 47.5 KB, 351,000 training samples, mini-batch size 1):

Layer Units Type Dropout L1 L2 Mean Rate Rate RMS Momentum Mean Weight Weight RMS Mean Bias Bias RMS
1 16 Input 0.00 %
2 50 Rectifier 0.00 % 0.000010 0.000000 0.001663 0.001707 0.000000 0.000248 0.187112 0.496939 0.102252
3 50 Rectifier 0.00 % 0.000010 0.000000 0.006981 0.011044 0.000000 0.000320 0.176389 0.968082 0.058068
4 2 Softmax 0.000010 0.000000 0.002517 0.001864 0.000000 0.024380 0.439828 0.000356 0.008649

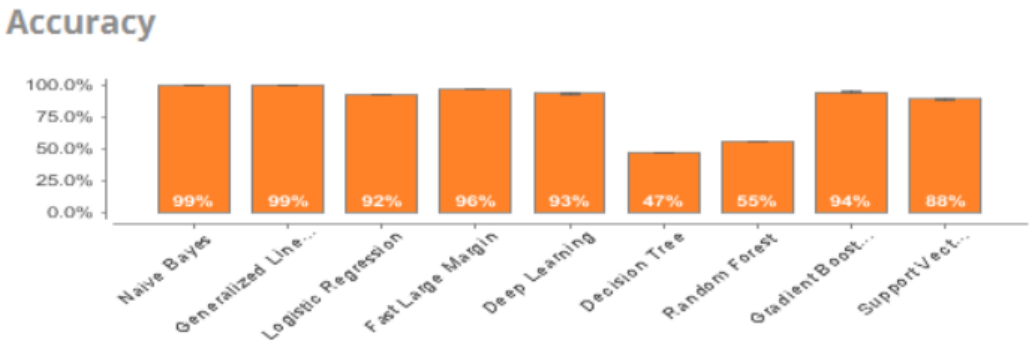
Scoring History:

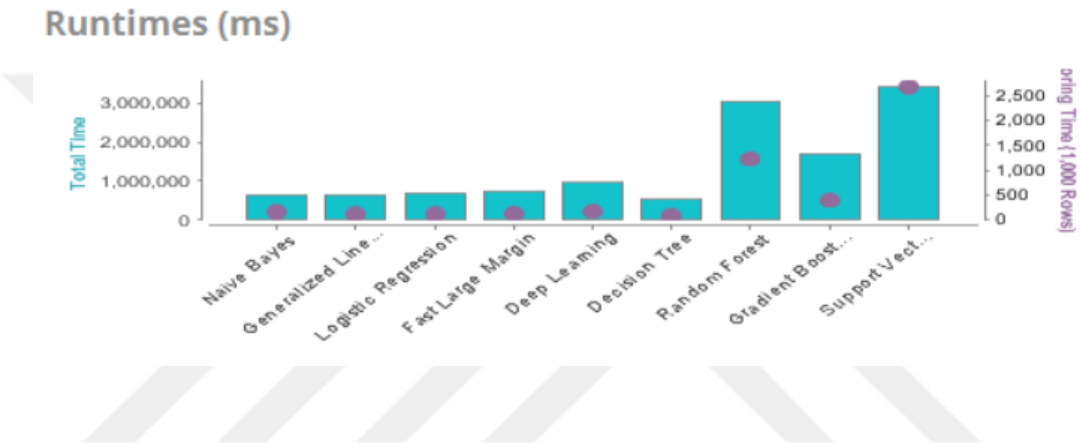
Timestamp	Duration	Training Speed	Epochs	Iterations	Samples	Training MSE	Training R^2	Training LogLoss	Training AUC	Training Lift	Training Classification Error
2019-09-30 20:14:52	0.000 sec		0.00000	0	0.000000	NaN	NaN	NaN	NaN	NaN	NaN
2019-09-30 20:14:54	1.655 sec	23046 rows/sec	1.00000	1	35100.000000	0.01094	0.95622	0.03591	0.99966	1.98655	0.01026
2019-09-30 20:14:56	3.218 sec	23549 rows/sec	2.00000	2	70200.000000	0.00600	0.97600	0.02099	0.99980	1.98655	0.00697
2019-09-30 20:14:57	4.744 sec	23926 rows/sec	3.00000	3	105300.000000	0.00533	0.97869	0.01845	0.99981	1.98655	0.00667
2019-09-30 20:14:59	6.236 sec	24269 rows/sec	4.00000	4	140400.000000	0.00686	0.97257	0.02201	0.99988	1.98655	0.00568
2019-09-30 20:15:00	7.704 sec	24562 rows/sec	5.00000	5	175500.000000	0.00454	0.98183	0.01506	0.99993	1.98655	0.00428
2019-09-30 20:15:02	9.149 sec	24837 rows/sec	6.00000	6	210600.000000	0.00534	0.97866	0.01675	0.99994	1.98655	0.00428
2019-09-30 20:15:03	10.575 sec	25081 rows/sec	7.00000	7	245700.000000	0.00414	0.98343	0.01357	0.99994	1.98655	0.00418
2019-09-30 20:15:04	12.002 sec	25279 rows/sec	8.00000	8	280800.000000	0.00504	0.97982	0.01606	0.99993	1.98655	0.00448
2019-09-30 20:15:06	13.393 sec	25498 rows/sec	9.00000	9	315900.000000	0.00365	0.98540	0.01210	0.99992	1.98655	0.00438
2019-09-30 20:15:07	14.778 sec	25687 rows/sec	10.00000	10	351000.000000	0.00264	0.98945	0.00911	0.99996	1.98655	0.00279

H2O version: 3.8.2.6-rm9.0.0

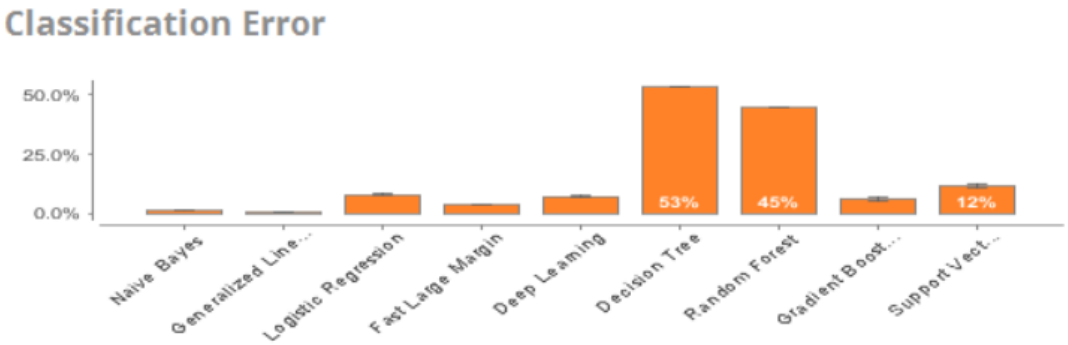
EK-J Ekstradan Sonuçlar

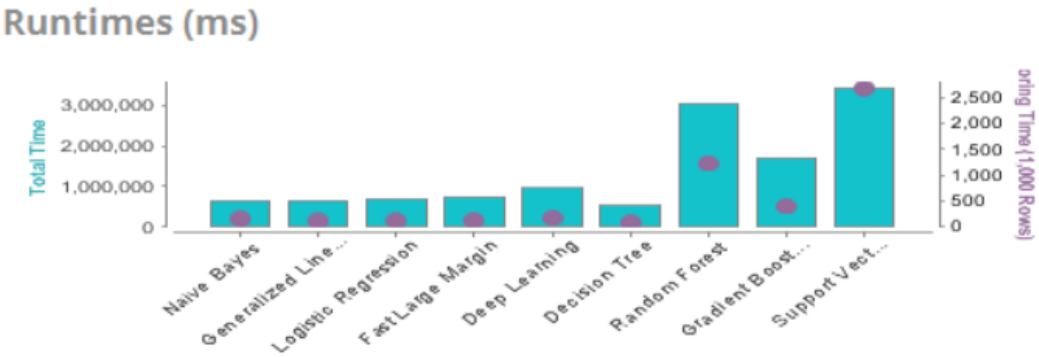
J1: Doğruluk (Accuracy) Bazında



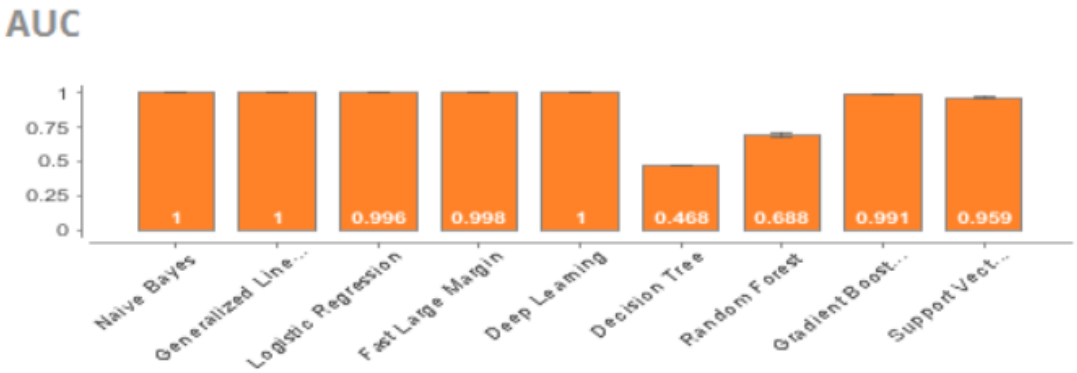


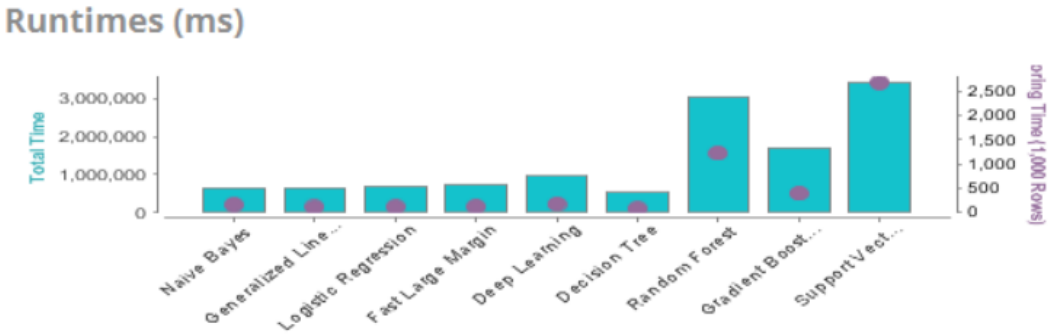
J2: Sınıflandırma Hatası (Classification Error) bazında



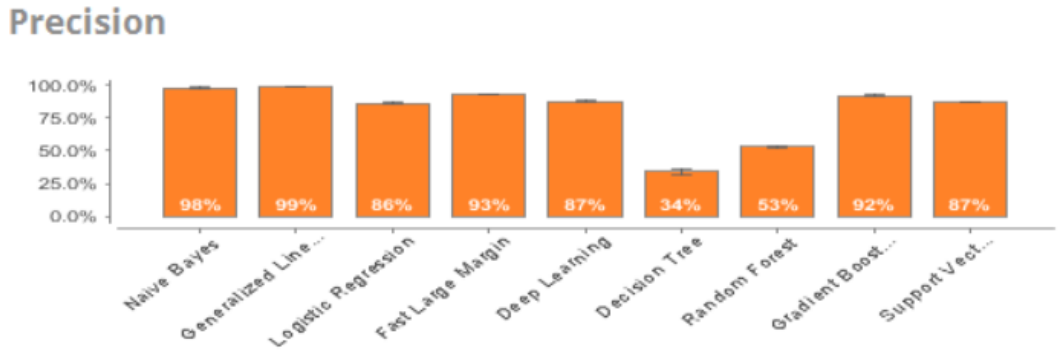


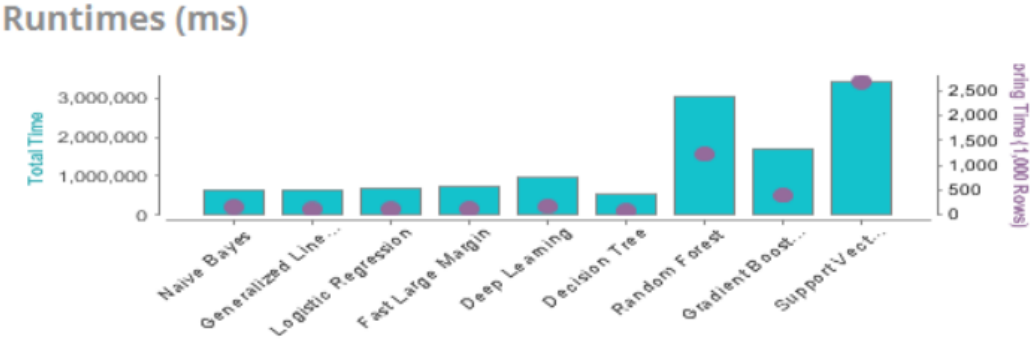
J3: Eğrinin Altındaki Alan (AUC) bazında



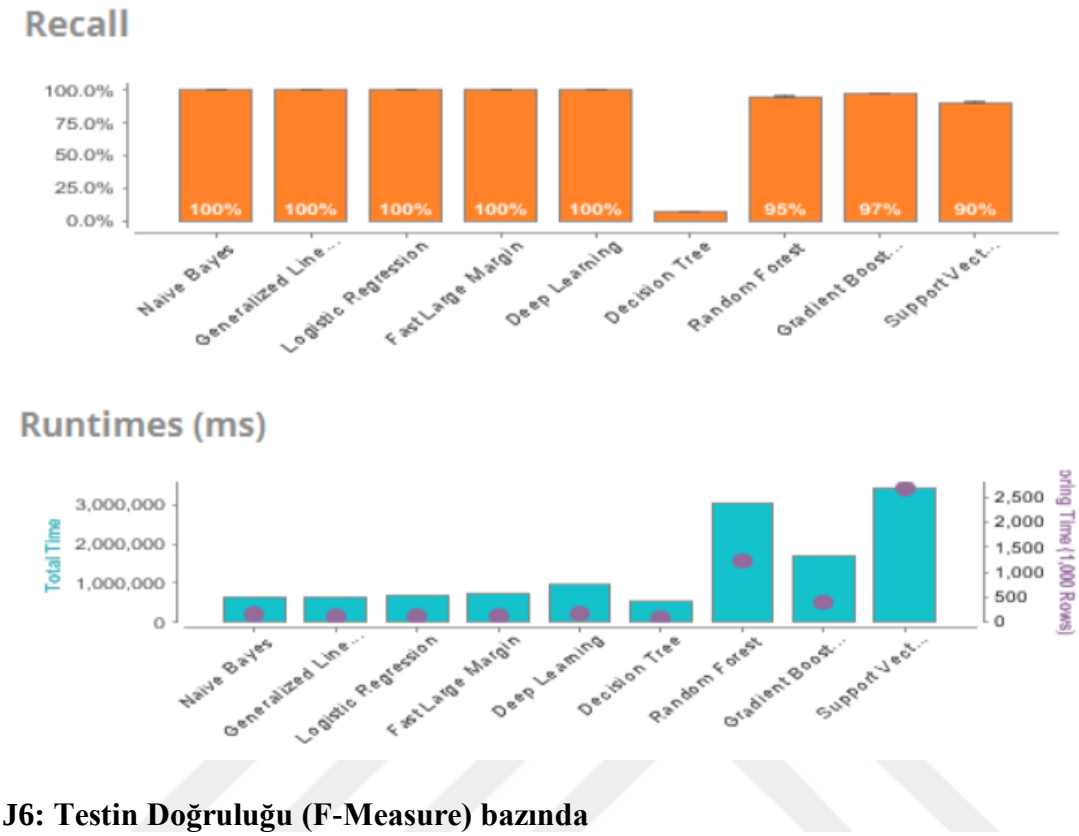


J4: Hassasiyet (Precision) bazında

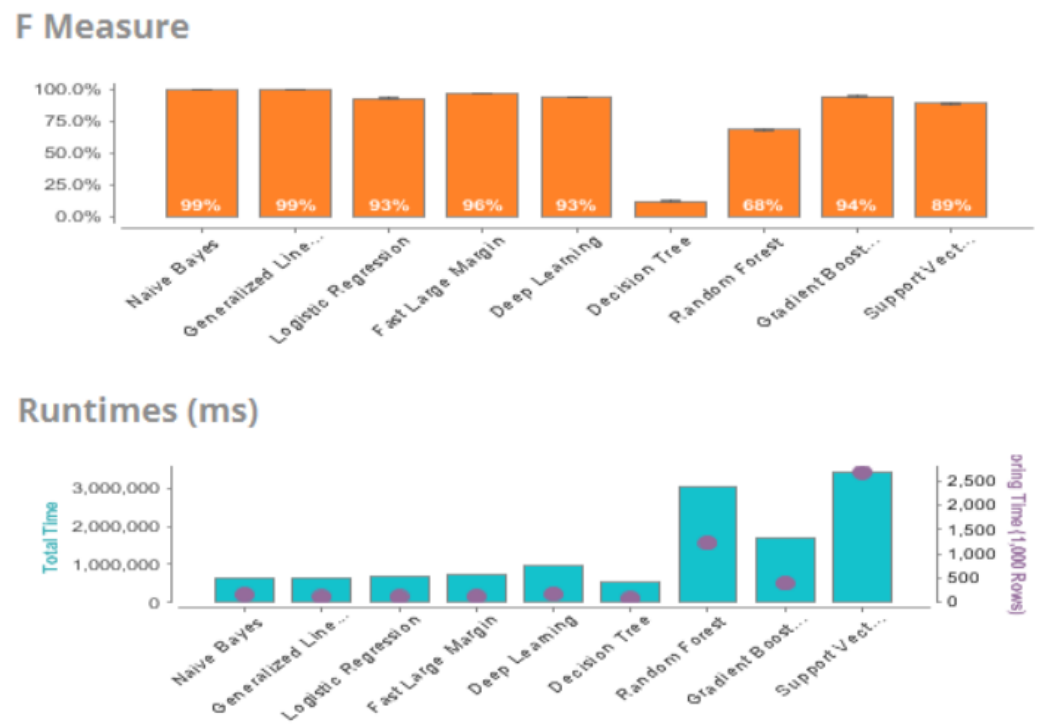




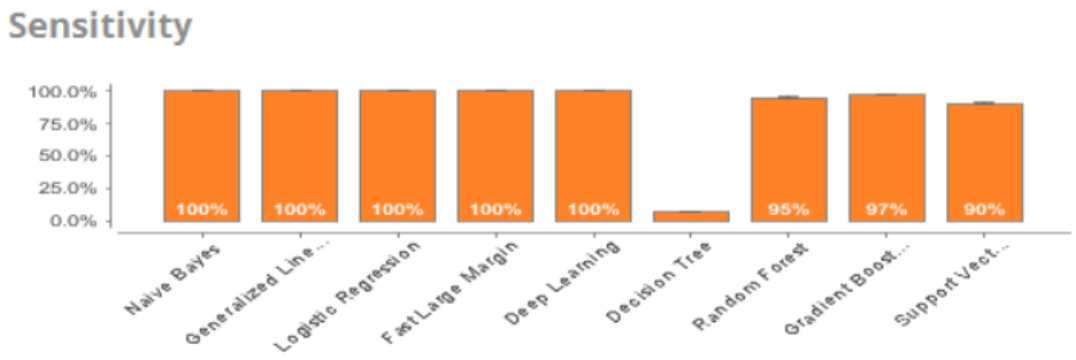
J5: Geri Çağırma (Recall) bazında

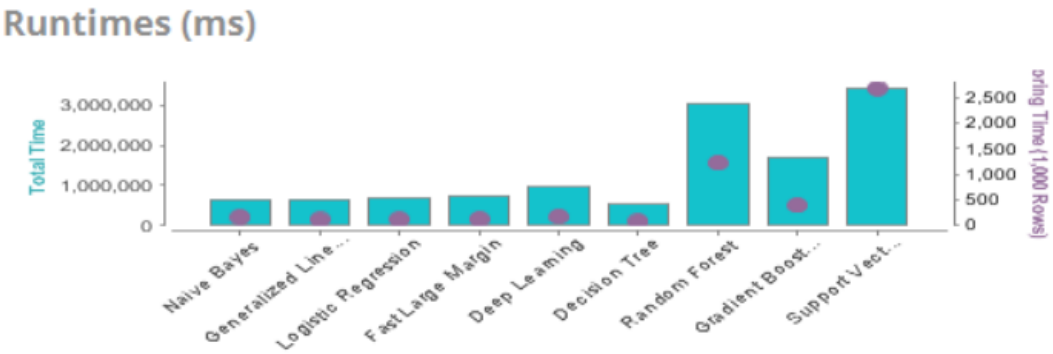


J6: Testin Doğruluğu (F-Measure) bazında

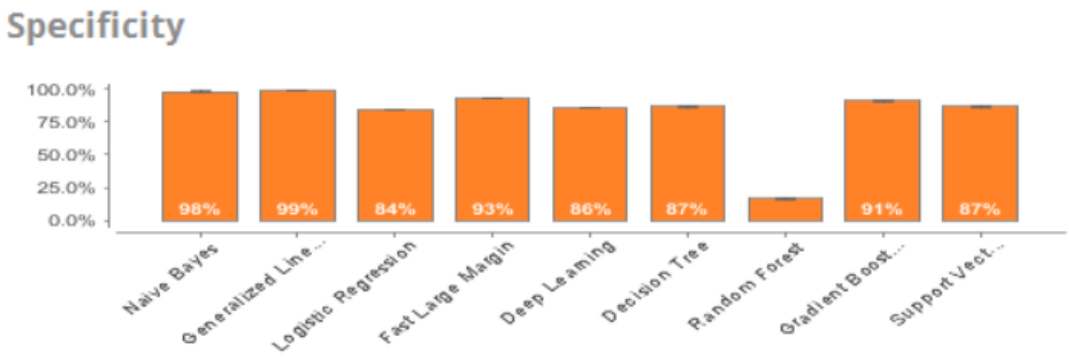


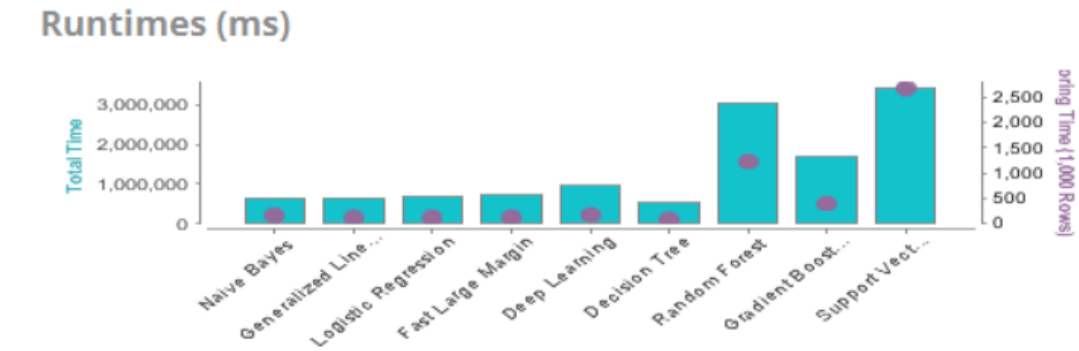
J7: Gerçek Pozitifler Oranı (Sensitivity) bazında





J8: Belirlilik (Specificity) bazında





ÖZGEÇMİŞ

ÖGRENİM DURUMU : Yüksek Lisans (Devam Etmekte)

- **LİSANS** : İstanbul Kültür Üniversitesi – Fen Edebiyat Fakültesi – Matematik&Bilgisayar (2008-2012)
- **YÜKSEK LİSANS** : İstanbul Aydın Üniversitesi - Bilgisayar Mühendisliği Ana Bilim Dalı - Bilgisayar Mühendisliği Programı

Serbest Yazılım Geliştirici (Mart 1998 – Kasım 2013)
Matematik Öğretmeni (Fatih Dersanesi)(Kasım 2013 – Mart 2014)
Yazılım Uzmanı (İdeasoft) (Mart 2014- Nisan 2015)
Yazılım Uzmanı (ÖZYURT A.Ş.) (Mayıs 2015 – Ekim 2015)
Yazılım ve Sistem Uzmanı (Perkotek) (Ekim 2015- Temmuz 2016)
Yazılım ve Entegrasyon Uzmanı (SETMODA) (Ağustos 2016 – Ekim 2017)
Software Developer & API & Integration Executive, Team Leader (Travelport Turkey, Travtech Solutions) (Ekim 2017-Aralık 2018)
Araştırma Görevlisi, Yazılım Mühendisliği(İngilizce) , İstanbul Aydın Üniversitesi (Aralık 2018-*)