T.C.
İSTANBUL ÜNİVERSİTESİ-CERRAHPAŞA
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

DOKTORA TEZİ

YAZILIM PROJELERİNİN YÖNETİMİNDE MALİYET TAHMİNİ İÇİN DERİN ÖĞRENME TABANLI YENİ BİR YAKLAŞIM

Senem OLGUN

DANIŞMAN
Doç. Dr. Tarık KÜÇÜKDENİZ

Endüstri Mühendisliği Anabilim Dalı

Endüstri Mühendisliği Programı

İSTANBUL-2020

Bu çalışma, 30.06.2020 tarihinde aşağıdaki jüri tarafından Endüstri Mühendisliği Anabilim Dalı, Endüstri Mühendisliği Programında Doktora tezi olarak kabul edilmiştir.

Tez Jürisi

Doç. Dr. Tarık KÜÇÜKDENİZ(Danışman)
İstanbul Üniversitesi-Cerrahpaşa
Mühendislik Fakültesi

Prof. Dr. Şakir ESNAF
İstanbul Üniversitesi-Cerrahpaşa
Mühendislik Fakültesi

Prof. Dr. Alp BARAY
İstanbul Üniversitesi-Cerrahpaşa
Mühendislik Fakültesi

Prof. Dr. Ferhan ÇEBİ
İstanbul Teknik Üniversitesi
İşletme Fakültesi

Doç. Dr. Emrah ÖNDER
İstanbul Üniversites-Cerrahpaşa
İşletme Fakültesi

20.04.2016 tarihli Resmi Gazete'de yayımlanan Lisansüstü Eğitim ve Öğretim Yönetmeliğinin 9/2 ve 22/2 maddeleri gereğince; Bu Lisansüstü teze, İstanbul Üniversitesi-Cerrahpaşa'nın abonesi olduğu intihal yazılım programı kullanılarak Lisansüstü Eğitim Enstitüsü'nün belirlemiş olduğu ölçütlere uygun rapor alınmıştır.

ÖNSÖZ

Tezimin hazırlanmasında ve akademik çalışmalarım esnasında yapmış olduğu katkılardan ötürü tez danışmanım Doç. Dr. Tarık Küçükdeniz'e.

Yapay Zeka konusuyla beni tanıştıran ve eğitimim boyunca deneyimine ve derin bilgi birikimine başvurduğum değerli hocam Prof. Dr. Şakir Esnaf'a.

Doktora eğitimim boyunca yönlendirmeleri ve yardımlarıyla yanımızda olan değerli hocam Prof. Dr. Alp Baray'a.

Akademik deneyimi ve pozitif enerjisiyle tezim boyunca desteği için değerli hocam Prof. Dr. Ferhan Çebi'ye,

İş hayatım boyunca sürdürdüğüm doktora eğitimimde desteğini esirgemeyen saygı değer Ceo'muz Tarkan Ersubaşı'na,

Bu zorlu sürecimde yanımda olan, bana motivasyon ve güç veren çok kıymetli arkadaşlarıma,

Ve son olarak hayatımın her döneminde olduğu gibi bu döneminde de sonsuz güvenle ve fedakarlıkla yanımda olan başta çok değerli annem, babam ve kardeşim Serkan Olgun olmak üzere tüm aileme teşekkürlerimi sunarım.

Haziran 2020 Senem OLGUN

İÇİNDEKİLER

ÖNSÖZ ..... **iv**
İÇİNDEKİLER ..... **v**
ŞEKİL LİSTESİ ..... **viii**
TABLO LİSTESİ ..... **ix**
KISALTMA LİSTESİ ..... **x**
ÖZET **1**
SUMMARY **3**
1. GİRİŞ **4**
1.1. KATKILAR: 5
1.2. TEZİN ANA HATLARI 5
2. GENEL KISIMLAR **8**
2.1. YAZILIM PROJE YÖNETİMİ 8
2.2. YAZILIM PROJELERİNDE EFOR TAHMİNİNİN ÖNEMİ 9
2.3. YAZILIM PROJELERİNDE EFOR TAHMİN YÖNTEMLERİ 13
2.3.1. Uzman Kararı: *14*
2.3.2. Algoritmik Yöntemler: *14*
2.3.3. Algoritmik Olmayan Modeller: *15*
2.4. MAKİNE ÖĞRENMESİ 15
2.4.1. Makine Öğrenmesi ve Yazılım Efor Tahmini Otomasyonu *17*
2.4.2. Literatürde Makine Öğrenmesiyle Yazılım Efor Tahmini *19*
2.5. DERİN ÖĞRENME VE YAPAY SİNİR AĞLARI 23
2.5.1. Derin Öğrenme ve Yapay Sinir Ağları *23*
2.5.2. Yapay Sinir Ağları ve Biyolojik Sinir Ağları *24*
2.5.3. Yapay Sinir Ağlarının Yapılarına Göre Sınıflandırılması *26*
2.6. KARAR AĞAÇLARI 29
2.6.1. Karar Ağaçlarının Yapısı ve Oluşturulması: *29*
2.6.1. Regresyon Ağaçları *31*
2.6.2. Model Birleştirme Yöntemleri *32*
2.6.3. Güçlendirilmiş Karar Ağacı Regresyonu *33*
2.7. VERİ MADENCİLİĞİ VE METİN MADENCİLİĞİ 35

2.7.1. Veri Madenciliği *35*
2.7.2. Veri Madenciliği ile Metin Madenciliği İlişkisi *36*
2.7.3. Metin Madenciliği *36*
3. MALZEME VE YÖNTEM **39**
3.1. VERİ HAZIRLIĞI 39
3.1.1. Metin ön işleme *39*
3.1.2. Öznitelik Karması: *41*
3.1.3. Kayıp Veri Temizliği *44*
3.2. HİPERPARAMETRE AYARLAMA 45
3.2.1. Manuel Hiperparametre Ayarlama *46*
3.2.2. Otomatik Hiperparametre Ayarlama: *46*
3.3. DERİN SİNİR AĞI HİPERPARAMETRELERİ 47
3.3.1. Aktivasyon Fonksiyonu: *47*
3.3.2. Katman sayıları: *51*
3.3.3. Veri seti büyüklüğü *53*
3.3.4. Öğrenme iterasyonlarının sayısı: *53*
3.3.5. Öğrenme Oranı: *54*
3.3.6. Momentum: *55*
3.3.7. Mini parti büyüklüğü(Mini batch size): *55*
3.3.8. Başlangıç ağırlığı ve ağırlık değişimleri: *56*
3.3.9. Kayıp Fonksiyonları *57*
3.3.10. Regülasyon(Regülarizasyon) *58*
3.4. DSA'DA HİPERPARAMETRE SEÇİMİ İÇİN DİKKAT EDİLMESİ GEREKEN DURUMLAR 61
3.4.1. Ağın Yerel Minimumda Salınımı *61*
3.4.2. Aşırı Uyum(Overfitting) *62*
3.4.3. Yetersiz Uyum(Underfitting) *64*
3.5. GÜÇLENDİRİLMİŞ KARAR AĞACI HİPERPARAMETRELERİ 66
3.5.1. Ağaç başına maksimum yaprak sayısı: *66*
3.5.2. Yaprak düğümü başına minimum örnek sayısı: *66*
3.5.3. Öğrenme oranı: *67*
3.5.4. Oluşturulan ağaç sayısı: *67*
3.6. REGRESYON DEĞERLENDİRME METRİKLERİ 67
3.6.1. Ortalama mutlak hata (MAE:Mean Absolute Error): *67*
3.6.2. Ortalama hata kare kökü (RMSE: Root Mean Squared Error): *67*

3.6.3. Göreceli mutlak hata (RAE: Relative Absolute Error): *68*
3.6.4. Göreceli hata karesi (RSE: Relative Squared Error): *68*
3.6.5. Determinasyon Katsayısı $r2$ *(COD: Coefficient of Determination):* *68*
4. BULGULAR **69**
4.1. VERİ SETİ 70
4.2. VERİ HAZIRLIĞI 73
4.2.1. Metin Ön İşleme *73*
4.2.2. Öznitelik Karması *74*
4.2.3. Kayıp Veri Temizleme *76*
4.3. HİPERPARAMETRE AYARLAMA 76
4.3.1. Derin Sinir Ağı Regresyon Modeli *76*
4.3.2. Güçlendirilmiş Karar Ağacı Regresyon Modeli *78*
4.4. UYGULAMA VE ANALİZ SONUÇLARI 78
4.4.1. Grid Arama Sonuçları *79*
4.4.2. Karşılaştırmalı Değerlendirme Sonuçları *82*
5. TARTIŞMA VE SONUÇ **87**
KAYNAKLAR **90**
ÖZGEÇMİŞ **99**

ŞEKİL LİSTESİ

Şekil 2-1 Makine Öğrenmesi Algoritmaları 16
Şekil 2-2 Temel insan sinir hücresi(Anderson ve diğ., 1992) 25
Şekil 2-3 İleri Beslemeli Sinir Ağları 27
Şekil 2-4 Radyal Temel Fonksiyonu Sinir Ağları 28
Şekil 2-5 Tekrarlayan Sinir Ağları 28
Şekil 2-6 Karar Ağacı Örneği 30
Şekil 2-7 Regresyon Ağacı Örneği 32
Şekil 2-8 Gradyan Güçlendirme Aşamaları 34
Şekil 3-1 Matematiksel modelleme ile elde edilen bir Yapay Sinir Hücresi 47
Şekil 3-2 Doğrusal Aktivasyon Fonksiyonu 48
Şekil 3-3 Sigmoid aktivasyon fonksiyonu 49
Şekil 3-4 Tanh aktivasyon fonksiyonu 49
Şekil 3-5 Düzeltilmiş Doğrusal Aktivasyon Fonksiyonu 50
Şekil 3-6 ReLU ve softplus aktivasyon fonksiyonlarının görselleştirilmesi 51
Şekil 3-7 Öğrenme Oranı ve Gradyan iniş 54
Şekil 3-8 Öğrenme sürecinde ağırlık değişimlerinin gösterimi (Nielsen, 2015) 57
Şekil 3-9 Ağın yerel minimumda salınımı 61
Şekil 3-10 Validasyon 63
Şekil 3-11 Erken Durdurma 63
Şekil 3-12 k-Katlı Çapraz Doğrulama yaklaşımının gösterimi 64
Şekil 3-13 Yanlılık - Varyans -Uyum Dengesi 65
Şekil 4-1 Uygulama Akışı Şeması 69
Şekil 4-2 Microsoft Azure ML, Metin ön işleme 74
Şekil 4-3 Karma Boyutu ve N-gram sayısı - Determinasyon Katsayısı Grafiği 75

TABLO LİSTESİ

Tablo 2-1 Yazılım efor tahmini için 10 yılda uygulanan bazı makine öğrenmesi teknikleri 18
Tablo 2-2 Biyolojik sinir hücresi ile yapay sinir hücresi arasındaki ilişki 25
Tablo 4-1 Veri seti istek sayısı tablosu 72
Tablo 4-2 Veri Seti Örneği 72
Tablo 4-3 Karma Tablosu Örnek Gösterim 75
Tablo 4-4 DSA Regresyon Modeli Grid Arama Hiperparametre Değerleri 77
Tablo 4-5 GKA Regresyon Modeli Grid Arama Hiperparametre Değerleri 78
Tablo 4-6 DSA Regresyon Modeli, Veri Seti Bazında Grid Arama Sonuçları 80
Tablo 4-7 GKA Regresyon Modeli, Veri Seti Bazında Grid Arama Sonuçları 81
Tablo 4-8 DSAR ve GKAR Karşılaştırmalı Değerlendirme Sonuçları, MAE, RMSE, RAE, RSE, COD 83

KISALTMA LİSTESİ

Kısaltmalar	Açıklama
SDLC	: Software Development Life Cycle - Yazılım geliştirme yaşam döngüsü
COCOMO	: Constructive Cost Model
PSO	: Particle Swarm Optimization - Parçacık Sürü Optimizasyonu
MMRE	: Mean Magnitude of Relative Error - Göreceli Hatanın Ortalama Büyüklüğü
PRED	: Prediction - Tahmin
MSE	: Mean Squared Error - Ortalama Hata Karesi
MRE	: Mean Relative Error - Ortalama Göreceli Hata
MAE	: Mean Absolute Error - Ortalama Mutlak Hata
YSA	: Yapay Sinir Ağları
ANFIS	: Adaptive Network Based Fuzzy Inference Systems - Uyarlamalı Ağ Tabanlı Bulanık Çıkarım Sistemi
SVR	: Support Vector Regression – Destek Vektör Regresyon
GPR	: Gaussian Process Regression – Gauss Süreç Regresyonu
CFS	: Correlation-based Feature Selection - Korelasyon Temelli Öznitelik Alt Seçimi
DXP	: Distributed Extreme Programming - Dağıtılmış Ekstrem Programlama
DSA	: Derin Sinir Ağları
DBN	: Deep Belief Networks - Derin İnanç Ağları
CNN	: Convolutional Neural Networks - Evrişimsel Sinir Ağları
RNN	: Recurrent Neural Networks - Tekrarlayan Sinir ağları
AID	: Automatic Interaction Detection - Otomatik Etkileşim Tespiti
CART	: Classification and Regression Tree - Sınıflandırma ve Regresyon Ağacı
GBM	: Gradient Boosting Machine – Gradyan Güçlendirme Makinası
MART	: Multiple Additive Regression Trees – Çoklu Katkı Regresyon Ağacı
CRISP-DM	: Cross-Industry Standard Process for Data Mining

SEMMA : Sample, Explore, Modify, Model, and Assess

KDD : Knowledge Discovery in Databases

MAR : Missing at random - Rastgele Eksik Veri

MNAR : Missing not at random - Rastgele Olmayan Eksik Veri

MCAR : Missing completely at random - Rastgele Tamamen Eksik Veri

RMSE : Root Mean Squared Error - Ortalama hata kare kökü

RAE : Relative Absolute Error - Göreceli mutlak hata

RSE : Relative Squared Error - Göreceli hata karesi

COD : Coefficient of Determination - Determinasyon Katsayısı

DSAR : Derin Sinir Ağı Regresyon

GKAR : Güçlendirilmiş Karar Ağacı Regresyon

KLOC : Kilos of Lines Of Code - bin kod satırı

LOC : Lines Of Code - kod satırı

ÖZET

DOKTORA TEZİ

YAZILIM PROJELERİNİN YÖNETİMİNDE MALİYET TAHMİNİ İÇİN DERİN ÖĞRENME TABANLI YENİ BİR YAKLAŞIM

Senem OLGUN

İstanbul Üniversitesi-Cerrahpaşa
Lisansüstü Eğitim Enstitüsü
Endüstri Mühendisliği Anabilim Dalı

Danışman : Doç. Dr. Tarık KÜÇÜKDENİZ

Yazılım projelerinin başarıyla yönetilebilmesi için proje maliyetinin doğru, tutarlı ve makul sürede tahmin edilmesi kritik önem taşımaktadır. Literatürde yazılım geliştirme projelerinde maliyet doğrudan efor ile ilişkilendirilmiştir. Günümüzde yazılım projelerinde efor tahmini büyük çoğunlukla insan yargısına dayalı ve manuel olarak yapılmaktadır. Bu çalışmanın amacı, bu sürecin bir sistem aracılığıyla otomatik olarak gerçekleştirilmesini sağlamaktır.

Bu tezde yazılım efor tahmini için iki güçlü makine öğrenmesi modeli karşılaştırılarak en düşük hata oranlarını veren model belirlenmiştir. Veri seti öncelikle bir dizi ön işleme sürecinden geçirilerek, öğrenme aşamasında daha verimli kullanılacak hale getirilmiştir. Yazılım efor tahmini aşamasında, Derin Sinir Ağı Regresyon Modeli ve Güçlendirilmiş Karar Ağacı Regresyon Modeli, 10 katlı Çapraz Doğrulama Yöntemi ile değerlendirilerek Ortalama Mutlak Hata değerleri üzerinden karşılaştırılmıştır. Karşılaştırma sonucunda Derin Sinir Ağı Regresyon modelinin, efor tahmininde çok daha iyi bir performans sergilediği gözlemlenmiştir.

Haziran 2020, 110 sayfa.

Anahtar kelimeler: Yapay Sinir Ağları, Derin Sinir Ağları, Yazılım Proje Yönetimi, Yazılım Efor Tahmini, Veri Madenciliği, Metin Madenciliği, Karar Ağacı, Regresyon

SUMMARY

Ph.D. THESIS

A DEEP LEARNING BASED NOVEL APPROACH FOR COST ESTIMATION IN SOFTWARE PROJECTS MANAGEMENT

Senem OLGUN

Istanbul University-Cerrahpasa
Institute of Graduate Studies
Department of Industrial Engineering

Supervisor : Assoc. Prof. Dr. Tarık KÜÇÜKDENİZ

In order to manage software projects successfully, it is critical to estimate project cost accurately, consistently and in reasonable time. In the literature, the cost is directly associated with effort in software development projects. Recently, effort estimation in software projects is mostly done manually and based on human judgment. The purpose of this study is to ensure this process to be performed automatically through a system.

In this thesis, two powerful machine learning models is compared for software effort estimation, and the model with the lowest error rates is determined. The data set has been made more efficient to use in the machine learning process after applying several pre-processing steps on the dataset. In the software effort estimation stage, the Deep Neural Network Regression Model and the Boosted Decision Tree Regression Model are evaluated with the 10-fold Cross Validation Method and compared over the Mean Absolute Error values. As a result of the comparison, it was observed that the Deep Neural Network Regression model performed much better in the effort estimation

June 2020, 110. pages.

Keywords: Artificial Neural Networks, Deep Neural Networks, Software Project Management, Software Effort Estimation, Data Mining, Text Mining, Decision Tree, Regression

1. GİRİŞ

Elektronik teknolojilerin kullanımının yaygınlaşmasıyla gün geçtikçe önem kazanan ve kapsamı genişleyen yazılımlar için proje yönetimi gerekli bir hale gelmiştir. Yazılım proje yönetiminde en önemli faktörlerden biri, projenin öngörülebilir olmasıdır. Projelerin gerek maliyet, gerekse işgücü planlaması bakımından mümkün olduğunca belirli hale getirilmesi, olası riskleri ortadan kaldırmaktadır. Yazılım eforu tahminleri, yazılım geliştirme çalışmalarının önemli bir parçasıdır ve fizibilite analizleri, teklif verme, bütçeleme ve planlama için gerekli girdileri sağlar. Bir yazılım ürününü geliştirmek için geliştirme zamanını doğru tahmin etmek, hayati öneme sahiptir. Çünkü yazılım endüstrisinde kaliteli yazılımlar için ciddi bir rekabet vardır. Gereken efordan çok farklı tahminler, başta maliyet ve kalite olmak üzere birçok hususta kayıplara neden olabilir. Günümüzde efor tahmini için kullanılan en yaygın yöntemler, insan yargısına dayalı yöntemlerdir. Ancak insan yargısı birçok koşuldan etkilenebildiği için yeterince güvenilir olmayabilir. Bunun yanı sıra insan yargısına dayalı yöntemlerin uygulama süreci, tahmin kalemleri arttıkça ciddi bir iş yüküne de neden olabilir.

Son yıllarda piyasanın hızlı değişen dinamikleri, yazılım proje yönetimi süreçlerinde, geleneksel yöntemlere nazaran çevik yöntemlere yönelimi oldukça artırmıştır. Çevik proje yönetimi metodolojisi uygulanan yazılım projelerinde eforun öngörülebilmesi için en yaygın kullanılan metrik, hikaye puanlarıdır. Günümüzdeki işleyişte genellikle bu tahminler, her bir istek için ilgili kişiler tarafından sezgiye dayalı bir şekilde manuel yapılmakta, sonrasında bu tahminler ilgili birimin yöneticisi tarafından kontrol edilmektedir. Ancak bu haliyle bu süreç, fazla insan kaynağı kullanmasına karşın, tutarlılığı ve sürekliliği düşük bir süreçtir.

Bu çalışmanın amacı, bir çevik yazılım projesinin yönetiminde eforun doğru ve hızlı şekilde öngörümlenmesi için derin öğrenme tabanlı bir efor tahmin yaklaşımı önermektir. Bu çalışmada geçmiş istek verilerinden öğrenerek hikâye puanlarını tahmin eden bir derin sinir ağı regresyon yaklaşımı önerilecek ve öncesinde veri seti farklı ön işleme süreçlerinden geçirilerek yenilikçi bir bakış açısı getirilecektir.

1.1. KATKILAR:

Bu tezle literatürde az sayıda çalışmasına rastlanan belli başlı alanlarda yeni yaklaşımlar getirilerek, literatüre farklı bakış açıları kazandırılmıştır. Bu tezin gerek literatüre, gerekse yazılım projelerine sağladığı önemli katkılar bulunmaktadır.
Bunlardan bazıları şöyledir;

- Bu çalışma ile insan yargısına dayalı yanlı ve tutarsız olma riski taşıyan tahminler yerine, geçmiş verilerle makine öğrenmesine dayalı çok daha tutarlı ve tekrarlanabilir bir tahmin yöntemi geliştirilmiştir.

- Ayrıca manuel olarak yapıldığı ve kontrol edildiği için oldukça fazla insan kaynağı harcayan efor tahmin sürecini kısaltarak kaynak tasarrufu sağlamıştır.

- Bir diğer katkı ise efor tahmin tutarlılığının artırılmasıyla, tahmin hatalarından kaynaklanan risklerin azaltılması olmuştur. Bu riskler şöyledir;
  Eforun gereğinden yüksek tahmin edilmesi durumunda,
  - rakiplerden gelen daha kısa sürede proje tamamlama taahhüdüne karşılık firmanın/ekibin rekabet gücünün azalması,

  Eforun gereğinden düşük tahmin edilmesi durumunda,
  - belirlenen tarihe yetiştirilmesine zorlanırsa yazılım ürününün kalitesinin düşmesi,
  - projenin olması gereken zamanda bitmesi sağlanırsa da, zaman ve maliyet kaybı ve bu sebepten doğan kaynak planlama sıkıntıları.

1.2. TEZİN ANA HATLARI

Bu tezde, yazılım proje yönetiminde kritik öneme sahip bir adım olan efor tahmin sürecinin en hızlı ve sağlıklı yapılabilmesi için derin öğrenme tabanlı bir yaklaşım sunulmuştur. Bu yaklaşımın etkisinin artırılabilmesi için öğrenme sürecinden önce veri, bir dizi hazırlık sürecinden geçirilmiştir. Uygulama için son yıllarda yazılım projelerinin büyük çoğunluğunda rastlanan çevik proje yönetim metodolojisine dayanan ve daha önce literatürde pek kullanılmamış 16 adet açık kaynaklı veri seti seçilmiştir.

Bu çalışma, derin öğrenme tabanlı bir yaklaşımı, çevik proje yönetiminde efor tahmini yapabilmek için, veri setini öznitelik çıkarımı dahil bir dizi veri hazırlık sürecinden geçirdikten sonra uygulamasıyla ve güçlendirilmiş karar ağacı regresyon yöntemiyle karşılaştırmasıyla literatüre yenilikçi bir çalışma kazandırmıştır.

Tezin Genel Kısımlar bölümünde yazılım proje yönetiminin amaçlarından, süreçlerinden ve yazılım projelerinde efor tahminin öneminden bahsedilerek, çevik projelerde efor tahmin süreci anlatılmıştır. Bu konu içerisinde, günümüzde çevik projelerde efor tahmini sürecinin büyük çoğunlukla manuel olarak gerçekleştirilmesinin yarattığı zorluklar ve tezde önerildiği gibi otomatik bir sistem aracılığıyla yapılması durumunda oluşacak avantajlar listelenmiştir. Devamında yaygın olarak kullanılan makine öğrenmesi algoritmaları şema halinde gösterilerek, son yıllarda literatürde yazılım projelerinde efor tahmini için uygulanan yöntemler ve örnekler açıklanmıştır. Ayrıca tezin uygulama aşamasında kullanılan Veri ve Metin Madenciliği, Derin Sinir Ağları ve Karar Ağaçları yöntemleri, en genel haliyle bu bölümde anlatılmıştır.

Tezin Malzeme ve Yöntem bölümünde, uygulama kapsamında kullanılan yöntemler sırayla açıklanmıştır. Öncelikle metin ön işleme sürecinde en sık gerçekleştirilen işlemler listelenmiştir. Daha sonra, verinin öğrenme sürecindeki etkinliğini artırmak açısından metin verilerini tam sayıya dönüştüren öznitelik karması yöntemi, temel kavramlarıyla beraber detaylı şekilde açıklanmıştır. Veri hazırlığı konusu, kayıp verilere uygulanabilecek yöntemlerin anlatımıyla tamamlanmıştır. Uygulamada veri hazırlığı sürecinden sonra hiperparametre ayarlama sürecine geçilmektedir. Bu nedenle bu aşamada hiperparametre ayarlama yöntemleri ve uygulamada kullanılan modellerin temel hiperparametreleri anlatılmıştır. Son olarak da regresyon modellerinde performans değerlendirme metrikleri listelenerek bir sonraki bölüme geçilmiştir.

Bulgular bölümünde, tezin uygulama süreci detaylı şekilde anlatılmıştır. Öncelikle uygulamada kullanılan veri seti hakkında bilgilendirme yapılmış, veri hazırlığı aşamasında uygulanan tüm adımlar listelenmiştir. Bu adımların arasında, öznitelik karması sürecinde kullanılacak değerlerin nasıl seçildiği de aktarılmıştır. Devamında gerçekleştirilecek hiperparametre ayarlama sürecinde, DSAR (Derin Sinir Ağları Regresyon) ve GKAR (Güçlendirilmiş Karar Ağacı Regresyon) modelleri için grid aramanın hangi hiperparametrelerle ve hangi değerler

opsiyon alınarak yapılacağı gösterilmiştir. Devamında 16 veri setinin her biri üzerine DSAR ve GKAR modelleri için ayrı ayrı 8'er farklı grid arama yapılmış; her bir grid arama için de ilgili veri setinin 10 farklı parçası test edilmiştir. Çapraz Doğrulama yöntemiyle test edilen bu 10 farklı parçanın hata değerleri ortalaması ilgili grid aramanın ortalama hata değeri olarak alınmış; her bir grid aramadaki en düşük hata değerini veren kombinasyon, ilgili veri setinde kullanılacak hiperparametre değerleri olarak belirlenmiştir. Belirlenen bu grid arama sonuçları her model için ayrı ayrı gösterilmiştir. Daha sonra, veri seti bazında belirlenen bu hiperparametre değerleri ilgili modellere uygulanmış ve sonuçlar 5 ayrı performans metriğine göre karşılaştırılmıştır.

Tartışma ve Sonuç bölümünde tezin amacı, katkıları, getirdiği yenilikçi yaklaşımlar ve tezin akışı özetlenmiş; uygulama esnasında karşılaşılabilecek zorluklar ve tezin bir sonraki adımında yapılabilecek çalışmalar aktarılmıştır.

2. GENEL KISIMLAR

2.1. YAZILIM PROJE YÖNETİMİ

Proje yönetimi, projeye özgü hedeflere yönelik bir sistem yönetimi yaklaşımına ulaşmak amacıyla proje yöneticisi ve proje organizasyonunun projeyi planlanma, organize etme, koordine etme, kontrol etme ve kaynak planlanma çabalarıdır.

Yazılım proje yönetimi, içine insanların, yazılım ürününün, planlama ve izleme gibi proje yönetimi sürecinin dahil olduğu faaliyetlerdir. Yazılım proje yönetiminin temel görevleri arasında projenin uygulama planının geliştirilmesi, kaynakların organize edilmesi, kalite yönetimi ve maliyet yönetimi yer almaktadır.

Yazılım proje yönetimi, yazılımın tanımlanması, geliştirmesi ve bakımını kapsar. Bakım süreci boyunca devam eden koruyucu ve destekleyici yazılım mühendisliği faaliyetleri de proje yönetimine dahildir. Başarılı bir yazılım mühendisliği organizasyonunda bir yazılım, üretim, uygulama geliştirme ve bakım görevleri kapsamında belirlenen kaynak, maliyet, çizelge ve kaliteye uygun olarak tamamlanır. Bu süreçte yazılım proje yönetiminin amacı planlanan çizelgelere, maliyetlere ve diğer gereksinimlere uyacak şekilde projeyi ilerletmek ve tamamlanmasını sağlamaktır (Li ve diğ, 2015). Dolayısıyla proje yönetimi sürecinin tanımlı bir başlangıcı ve sonu vardır. Proje bittiğinde proje ekipleri başka projelere geçer.

Proje yönetimi birçok değişken içermektedir. Bu nedenle bir projeyi yönetmenin tek bir yolu bulunmamaktadır. Aynı proje farklı metodolojilerle yönetilebilir. Proje yönetim metodolojisi, bir diğer deyişle yazılım süreç modelleri, projelerin etkin şekilde yönetilmesi amacıyla oluşturulmuş prosedürler olarak tanımlanabilir. Projenin amacına ve hedeflerine göre geliştirilmesi için seçilen proje yönetim süreci veya metodolojisi yazılım geliştirme modelini oluşturmaktadır.

Yazılım geliştirme yaşam döngüsü (SDLC - Software Development Life Cycle), bir yazılım geliştirme projesinin aşamalarında gerçekleştirilen faaliyetleri açıklayan bir çerçevedir. Yazılımın geliştirilmesi ve bakımı için gereken tüm faaliyetleri kapsar. Ürünler, teslim edilen proje kapandıktan çok sonra da var olmaya devam eder. SDLC, üretim sonrası da ürünü

destekleyen bir çerçevedir. SDCL, bir sistemin yaşamında tam bir çember şeklindedir. SDLC için uluslararası standart ISO / IEC 12207'dir.

Yazılım geliştirme yaşam döngüsü için, her biri belirli hedeflere yönelik olarak geliştirilmiş çeşitli proje yönetim metodolojileri vardır. Başlıca proje yönetim metodolojileri aşağıda listelenmiştir (Stoica ve diğ, 2013).

- Şelale modeli;
- V modeli;
- Artımlı model;
- RAD modeli;
- Çevik model;
- Yinelemeli model;
- Sarmal model.

Her proje kendi içinde değerlendirilmeli ve en uygun metodoloji belirlendikten sonra çalışmalara başlanmalıdır. Bir yazılım projesinde yazılım proje yönetim metodolojisinin doğru seçilmesi, proje yönetiminin başarısını doğrudan etkilemektedir.

Günümüzde yazılım geliştirme alanındaki en popüler proje yönetimi yaklaşımlarından biri Çevik modeldir. Sektörler arası *State of Agile* araştırmasına göre, anket yapılan 1.319 şirketin %97'si çevik geliştirme yöntemlerini uygulamaktadır (Versionone, 2019). Bu çalışmada, yazılım sektöründe proje yönetimi alanındaki bu eğilim göz önünde bulundurularak Çevik proje yönetiminde efor tahmini üzerine bir yaklaşım sunulmuştur.

2.2.YAZILIM PROJELERİNDE EFOR TAHMİNİNİN ÖNEMİ

Tahmin yazılım geliştirme yaşam döngüsünün çok önemli ve ayrılmaz bir parçasıdır. Efor ve maliyet tahminleri olmadan yazılım geliştirilemez. Mümkün olduğunca doğru tahmin yapmak önemlidir (Owais ve Ramakishore, 2016). Yüksek fiyatlandırma da düşük fiyatlandırma da yazılım geliştiren işletmeye zarar verebilir (Bedi ve Singh, 2010).

Yazılım efor tahmini, bir yazılımı geliştirmek veya sürdürmek için gereken doğru eforu belirleme sürecidir. Yazılım mühendisliğinde hala tam olarak çözülmemiş olan önemli bir

problemdir. Efor tahminleri, yazılım geliştirme için gereken süreyi hesaplamak üzere yazılım mühendisliğinin ilk aşamalarında yapılır.

Yazılım efor tahmini, yazılım geliştirme sürecinin iyileştirilmesinde önemli bir rol üstlenir. Yazılım geliştirmedeki maliyetin kritik yönü insan eforudur. Bu nedenle çoğu maliyet tahmin tekniği bu konu üzerinde yoğunlaşmaktadır. Yazılım efor tahmini, bir yazılım geliştirme projesinin zamanında, bütçe dahilinde ve gereksinimleri karşılayacak kalitede üretimini hedefleyen adımlardan biridir. Çoğu geliştiricinin en büyük endişelerinden biri, tahminin geliştirme sürecinin erken bir aşamasında yapılmasından kaynaklanan belirsizliğidir. Bu belirsizlikten doğan risk, projeye tahsis edilecek kaynak açısından çok maliyetli olabilecek bir yazılım projesi için tahmin hatalarının büyüklüğü arttıkça artmaktadır (Arbain ve diğ., 2019).

Yazılım projelerinin eforunu tahmin etme hassasiyeti ve güvenilirliği, yazılım şirketlerinin rekabet gücü açısından çok önemlidir. Yazılım kullanımı günümüzde sürekli olarak artmaktadır. Yazılım firmalarının rekabet gücünü garantilemek için yüksek kalitede ve zamanında yazılım üretmeleri gerekmektedir. Yüksek rekabet ortamı nedeniyle yazılım endüstrisi, firmaları zamanında ve bütçeye uygun şekilde yazılım projeleri üretmeye zorlamaktadır. Artan planlama ve proje yönetimi ihtiyacı yazılım proje yöneticilerinin giderek daha fazla dikkat ve kontrolünü gerektirmektedir. Efor tahmini, yazılım planlama ve yönetiminde en önemli adımlardan biridir (Braga ve diğ, 2007).

Tüm proje yönetim süreçleri proje planlamasıyla başlamaktadır. Üretilen proje planları, proje çalışmalarını etkin bir şekilde koordine etmek ve projenin başarıyla uygulanmasını sağlamak için en önemli araçtır. Maliyet, zamanlama gibi projenin başarısını kısıtlayabilecek fonksiyonlar için etkili bir düzenleme yapabilmek ancak etkin bir proje planlaması ile mümkündür (Zhang ve diğ.,2006).

Yazılım proje yöneticilerinin karşılaştığı belirgin zorluklar arasında doğru ve tutarlı tahminde bulunulması gelmektedir. Proje yöneticileri kısıt yoğunluğu, zaman planlaması, efor gereksinimleri ve proje maliyetleri gibi proje sürecinde etkisi olan faktörleri öngörmek üzere görevlendirilir. Bu görevdeki yöneticilere, kapsamlı araştırmalar yapmak ve çok sayıda tahmin tekniklerini değerlendirmek destek olacaktır (Macdonell ve Shepperd, 2003).

Yazılım efor tahmini yapmak, temel olarak iki nedenden dolayı oldukça zordur. Birinci neden yazılımın soyut, elle dokunulamayan, alışılmış fiziksel ürün tanımının dışında olmasıdır. İkinci neden ise yazılım geliştirme işinin fiziksel bir işten ziyade entelektüel bir iş olmasıdır. Yazılım başlangıçları kolaydır ancak yazılım büyüklüğü arttıkça iş gücü tahmininde süreç zorlaşmaktadır. Birkaç bin satıra yakın büyüklükteki bir programı bir haftada yazmak mümkündür. Ancak daha sonra program büyüdükçe hız yavaşlar. Bu program birkaç on bin satıra ulaştığında bir satır ilave etmenin bedeli birkaç günlük, belki de birkaç aylık efordur. Dolayısıyla eklemenin oluşturacağı yan etkileri takip etmek zorlaşmıştır (Soner, 2014). Yazılım geliştirme endüstrisinde dinamik olarak dalgalanan teknoloji ortamı da efor tahminini ayrıca kafa karıştırıcı hale getirmektedir (Rijwani ve Jaini, 2016).

Diğer yandan kaynak gereksinimlerinin doğru ve tutarlı bir şekilde tahmin edilmesi, yazılım projelerinin etkin yönetiminde oldukça önemli bir bileşen oluşturmaktadır. Son 20 yıldaki kapsamlı araştırmalara rağmen, etkili kaynak tahmini konusunda yazılım toplulukları halen tartışmaktadır.

Bir *Standish Group's Chaos* raporunda, analiz edilen yazılım projelerinin %66'sı öngörülen bütçenin üzerinde veya gecikmeli olarak teslim edilmiştir veya daha kötüsü bitmemiştir. Bu gibi başarısızlıkların başlıca nedenlerinden biri, projelerin eforlarının yanlış tahmin edilmesidir. Dolayısıyla, bu tahminlerin doğruluğunu iyileştirmek için yeni yöntemleri araştırmak önem taşımaktadır (Braga ve diğ, 2007).

Güvenilir olmayan tahminler, proje başarısızlığının başlıca nedenlerinden biridir. Binlerce Bilgi Teknolojileri uzmanının yanıtlarından oluşan 2007 Comp TIA anketi, bir bilgi teknolojileri projesinin başarısızlık nedenlerinden dörtte üçünün, kötü tahminden kaynaklandığını ortaya koymaktadır (Ananthakumar ve diğ, 2016).

Çoğu yazılım projesinin efor tahmini yalnızca insan yargısına dayanmaktadır. İnsan muhakemesine dayanan efor tahmin teknikleri arasında, grup tahminine dayanan yöntemler en popüler olanlarıdır.

Projelerde kullanılan proje yönetim metodolojisine göre, efor tahmin metrikleri ve süreçleri değişkenlik göstermektedir. Bu çalışma Çevik projeler üzerine gerçekleştirildiği için çevik projelerde efor tahmin süreci aşağıda anlatılacaktır.

Çevik Projelerde Efor Tahmin Süreci:

Uygulanan proje yönetim metodolojisine göre proje eforunu hesaplamak için kullanılan metrikler ve yöntemler farklılık göstermektedir. Hikâye puanı, çevik proje yönetimindeki eforun tahmin edilmesini sağlayan bir ölçü birimidir. Bir geliştirme isteğini(görevini) tamamlamak üzere çevik ekipler tarafından sarf edilecek eforu tahmin etmek için kullanılır. Belirli bir isteği tamamlamak için yapılması gereken iş miktarının ölçülmesini sağlar. Hikâye puanları temelde verilen bir isteğin geliştirme ekibi için ne kadar zor olduğunun bir göstergesidir. İstek eforlarına karşılık gelecek hikâye puanları üzerine ekip karar birliğine varır. Bu durumda, farklı ekipler aynı istek için farklı tahminler sağlayabilir.

Çevik gelişim kapsamında hikâye puanları genellikle planlama pokeri oturumları adı verilen yapılandırılmış grup toplantıları aracılığıyla atanır. Toplantı öncesi Fibonacci dizisine dayalı bir kart destesi temin edilir. Fibonacci dizisi (örneğin 0, 0.5, 1, 2, 3, 5, 8, 13, 20, 40, 100, ∞) her bir sayı kendinden önceki iki sayının toplamı olacak şekilde artan sayılardan oluşur. Bu sayıların aralarındaki farkın giderek artmasının bir nedeni, tahmin büyüdükçe kesinliğinin zor olması ve hata miktarının yükselmesidir. Diğer bir nedeni de, geliştiricileri orta değerden uzaklaştırarak fikir ayrılıklarının artmasını ve bu sayede farklı fikirlerin tartışılabilir olmasını sağlamaktır.

Öncelikle ilk istek hakkında ekibe bilgi verilir. İlk isteğin tahmininden önce geliştiriciler, kendi görüşüne en yakın olan hikâye kartını seçer ve poker oyunu başlar. Geliştiriciler seçtiği kartı oturum yöneticisinin işaretine kadar diğer geliştiricilere göstermez. İşaret sonrası herkes kartlarını açar. Oturum yöneticisi çoğunluğun seçtiği kartları ve çoğunluğun dışında kalan kartları belirler. Ardından bu kartlar arasından en düşük ve en yüksek kartı seçen geliştiricilerin fikirlerini açıklaması ister. Böylece bu görüşler ekip içerisinde tartışmaya açılır. Süreç, ekibin ilgili tahminde fikir birliğine varmasıyla tamamlanır. Bu döngü, diğer istekler için de devam eder.

Bu toplantılar büyük ölçüde insan yargısına dayanır. Uygulama geliştiriciler isteği ne kadar iyi anlarsa, o kadar sağlam bir tahmin sağlayabilirler. Ancak insan yargısını etkileyebilecek farklı unsurlar da bulunmaktadır. Bunlardan bazıları aşağıda listelenmiştir;

- İnsanlar genellikle iyimser tahminlerde bulunur. Bu yanlılık, grup etkileşimlerinde daha da belirginleşerek fazla iyimser tahminlere neden olabilir.
- Geliştiriciler tahminlerini ya da yorumlarını aktarırken, proje yöneticisinden, üst düzey geliştiricilerden veya toplantıdaki baskın kişilerden etkilenebilir.

Tahmin sürecinde insan yargısına dayalı bu ve benzeri sorunları çözebilmek için otomatik bir makine öğrenimi sistemi kullanılabilir (Porru ve diğ, 2016).

2.3.YAZILIM PROJELERİNDE EFOR TAHMİN YÖNTEMLERİ

Yazılım geliştirme eforlarını tahmin etmede ilk yaklaşımlar 1960'ların sonlarında ortaya atılmıştır ve uzman yargısına dayanmaktadır. Bu yaklaşıma göre bir uzman, efor tahmininde bulunurken daha önceki deneyimini kullanmaktadır. Bu yaklaşımın tahmin süreçlerinde bir dizi farklı varyasyon mevcuttur. Örneğin birkaç deneyimli geliştiricinin bağımsız tahminlerinin medyanı nihai efor tahmini olarak kullanılabilir. Uzman odaklı yaklaşım, şirketlerde hâlâ yaygın olarak kullanılmaya devam ederken, objektif bir zemin hazırlama konusunda da dezavantajlara sahiptir. Çünkü bu yaklaşımda tahmin sürecinin anahtarı uzmanlardır ve bu durum ek riskler içermektedir (Dejaeger ve diğ, 2012).

Yazılım efor tahmin modelleri, 30 yıldan uzun süredir gündemdeki bir çalışma konusudur. Bu konuda son on yılda birçok model geliştirilmiştir. Buna rağmen son istatistikler, bu alandaki çalışmaların yazılım geliştiricilerin gereksinimlerini karşılamadığını göstermektedir (Barati ve diğ., 2016).

Yazılım proje yönetiminde eforu etkileyen kritik değişkenleri anlamak ve kontrol etmek büyük önem taşımaktadır. Bu amaçla birçok tahmin yöntemi geliştirilmiştir. Yazılım efor tahmin yöntemleri üç ana kategoriye ayrılır.

2.3.1. Uzman Kararı:

Bu yöntemde yazılım geliştirme süreçleri uzmanı geçmiş deneyimlerine dayanarak yazılım geliştirme parametrelerini tahmin eder. Bu tür tahminlerin doğruluğu, büyük ölçüde uzmanın yeteneğine ve yeni bir projenin uzmanın deneyimiyle ne ölçüde eşleştiğine bağlıdır. Son zamanlarda yapılan deneyler, yazılım geliştirme eforu üzerine uzmanların yargıya dayalı tahminlerinde yüksek derecede tutarsızlık olduğunu göstermektedir.

Mahmood ve diğ. tarafından 2019'da bu alanda kapsamlı bir çalışma yapılmıştır. Bu çalışmada 2000-2019 döneminde yayınlanan toplam 34 adet vaka noktası ve uzman yargı temelli tahmin teknikleri çalışmasının sistematik bir değerlendirmesi yapılmıştır. Bu çalışmada uzman kararı tekniğinin en çok yazılım sektöründe kullanıldığı ve en sık kullanılan veri setlerinin yazılım endüstrisi projelerinden toplandığı tespit edilmiştir. Çalışmanın doğruluk oranlarına göre, sonuçlarda çelişkilerin olduğu gözlenmiştir ve herhangi bir yazılım efor tahmininde bu yöntemi uygulamanın pek umut verici sonuçlar vermeyeceği düşünülmüştür.

2.3.2. Algoritmik Yöntemler:

Algoritmik yöntemler, bağımsız değişkenleri, bağımlı değişkenlerle (efor, maliyet gibi) ilişkilendirmek için matematiksel formüller içeren yaklaşımlardır. Bu yöntemlerde belirli parametreler kullanılarak bağımlı değişken çıktı olarak elde edilir. Algoritmik tahmin üzerine en iyi bilinen modellerden biri, Barry Boehm tarafından geliştirilen, tahmin için temel bir regresyon formülünü kullanan COCOMO(Constructive Cost Model) modelidir. Bir diğer yöntem ise mevcut projeyle geçmiş projeler arasındaki benzerlikleri kullanan anoloji tabanlı tahmin yöntemidir (Humayun ve diğ, 2012).

Geçmişte, efor ve maliyet modelleme çalışmalarında algoritmik yöntemlere çok sık rastlanmaktaydı. Araştırmacılar, geçmiş projelerdeki verileri ve bunlardan üretilen denklemleri inceleyip, gelecekteki projeler için efor ve maliyet tahmininde bulunmaktaydı. Ancak hızla değişen iş ve yazılım endüstrisi, yeni geliştirme metodolojilerini, yeni teknolojileri ve yeni araçları beraberinde getirmiştir. Algoritmik modellerin yeni koşullara uyum sağlamada esnekliğinin olmaması nedeniyle algoritmik modellerin hızla modası geçmiştir (Lin ve diğ, 2010).

Algoritmik modeller matematiksel hesaplama yöntemini uygularken; algoritmik olmayan modeller uzman sistemler ya da bulanık mantık, yapay sinir ağları gibi makine öğrenme tekniklerini kullanır. Proje yöneticilerinin tahmin metodolojilerinde kararsız olması durumunda, proje yönetiminin etkinliği tehlikeye atılacaktır (Ananthakumar ve diğ, 2016).

2.3.3. Algoritmik Olmayan Modeller:

Son yıllarda araştırmacılar, doğru efor tahmini için yapay zeka tabanlı teknikler kullanarak yeni bir yaklaşım keşfetmeye odaklanmışlardır. Bu yaklaşım, yazılım geliştirme projelerinin eforlarını tahmin etmek için yapay zekanın alt alanı olarak makine öğrenmesini kullanır.

Makine Öğrenmesinin tanımı, algoritmaları, literatür araştırması ve efor tahmini üzerine kullanımı hakkında detaylı bilgi bir sonraki bölümde anlatılacaktır.

2.4.MAKİNE ÖĞRENMESİ

Makine öğrenmesi, makinelerin deneyime dayalı öğrenme yolları ile ilgilenen bir bilimsel alandır. Öğrenme, sözcüğün en geniş anlamıyla akıllı denilen bir oluşumun temel özelliğidir. Makine öğrenmesinin amacı, öğrenebilen bilgisayar sistemlerinin inşasıdır.

Makine öğrenmesi, çeşitli matematiksel yöntemler kullanarak mevcut verilerden bilgi edinen ve edindiği bu bilgi ile tahminler yapabilen yöntemler bütünü olarak ele alınabilir. Makine öğrenmesi algoritmaları, eldeki ham veriyi kullanmadan önce, bu verileri temsil eden daha anlamlı veriler çıkartır. Örneğin, eğitim aşamasında, algoritmada kullanılan matematiksel metotlarla mevcut eğitim verileri işlenir ve ortaya bir model çıkartılır. Artık bu model yardımı ile herhangi bir test verisi hakkında tahminde bulunulabilir.

Makine öğrenmesi algoritmaları temelde üç geniş kategoriye ayrılır. Bunlar şöyledir; öğrenme sisteminin etiketli eğitim verilerinden bir fonksiyona girdiği danışmanlı öğrenme, öğrenme sisteminin etiketsiz verilerin yapısını çıkarmaya çalıştığı danışmansız öğrenme ve sistemin dinamik bir ortamla etkileşime girdiği takviyeli öğrenme (Kavakiotis ve diğ, 2017).

- Danışmanlı(Supervised) Öğrenme
- Danışmansız(Unsupervised) Öğrenme
- Takviyeli(Reinforcement) Öğrenme

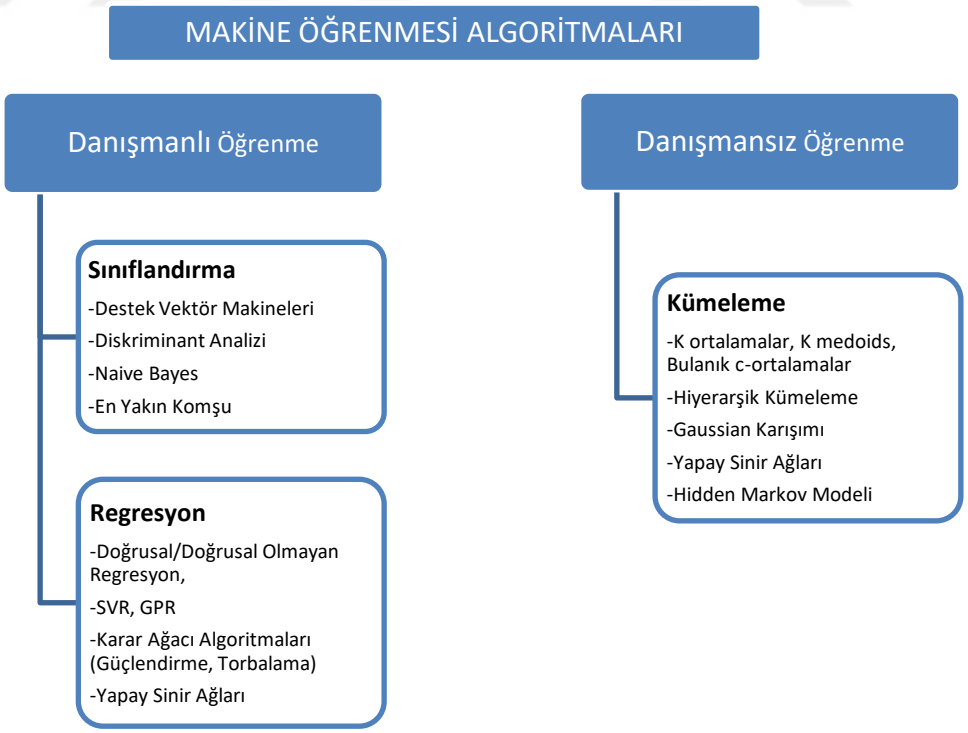
Danışmanlı öğrenmede, öğrenme sürecinde bir öğretmen bulunur. Bu süreçte girdi değerleri ve beklenen çıktı ağa zaten sunulmuştur. Bu yöntemlere örnek olarak regresyon ve sınıflandırma algoritmaları verilebilir.

Danışmansız öğrenmede, öğrenme sürecinde bir öğretmen yoktur. Beklenen çıktı ağa sunulmaz. Sistem, girdi modelindeki yapısal özelliklerin keşfedilmesi ve uyarlanması ile kendi kendine öğrenir. Bu yöntemlere örnek olarak kümeleme algoritmaları verilebilir.

Takviyeli öğrenme yönteminde bir öğretmen vardır. Ancak beklenen çıktı ağda mevcut değildir. Sadece çıktının doğru ya da yanlış olduğu gösterilir. Bu yöntemde yanlış çıktı için ceza, doğru çıktı için ödül verilir.

Bu yöntemler arasında Danışmanlı ve Danışmansız yöntemler, ağın eğitiminde kullanılan en popüler yöntemlerdir (Panchal and Panchal, 2014).

Yaygın olarak kullanılan makine öğrenmesi teknikleri Şekil 2.1'de gösterilmiştir (https://www.mathworks.com/discovery/machine-learning.html).



Şekil 2-1 Makine Öğrenmesi Algoritmaları

Hangi tekniğin hangi veri setinde daha doğru sonuç verdiğini belirlemek zordur. Literatürde makine öğrenim teknikleri alanında çok fazla araştırma yapılmıştır. Bu araştırmalar, makine öğrenmesi yöntemlerinin özellikle yazılım geliştirme projelerinde geleneksel modellere kıyasla daha yeterli tahmin modelleri sağlayabildiğini göstermektedir (Humayun ve diğ, 2012).

Bu çalışmada girdi değerleri ve beklenen çıktı değeri ağa sunulmuştur. Dolayısıyla danışmanlı öğrenme gerçekleştirilecektir. Beklenen çıktı değeri hikaye puanlarından oluşmakla birlikte sayısal değerler içerdiği için bir regresyon modeli ile tahmin gerçekleştirilecektir. Tahmin yöntemleri arasından Derin Sinir Ağı Regresyon Modeli ile Güçlendirilmiş Karar Ağacı Regresyon Modeli karşılaştırılmıştır.

2.4.1. Makine Öğrenmesi ve Yazılım Efor Tahmini Otomasyonu

Yazılım efor tahmininin doğruluğu, öğrenmenin tahmin sürecinde ne kadar veri kullanılarak gerçekleştiğiyle yakından ilişkilidir. Ancak insan beyni bu açıdan kısıtlı kalmaktadır. Efor tahmininin manuel olarak yapılması veri miktarını sınırlandırabileceğinden, bu süreçte otomatik bir sistem kullanmanın tahmin doğruluğuna etkisi büyüktür. Çevik projelerde ekipler küçük iterasyonlar (sprintler) ile çalıştıkları için bir proje boyunca birkaç yüz istek üretilebilir. Otomatik bir sistem kullanılarak uygulama verisinin miktarı artırıldığında tahminlerin ekip için daha tatmin edici hale geldiği bilinmektedir (Porru ve diğ, 2016).

Yazılım efor tahmini sürecinde otomatik bir sistem kullanmanın diğer başlıca avantajları aşağıda listelenmiştir.

- Otomatik sistem proje başlangıcından bu yana proje hakkında ayrıntılı bilgiye sahiptir ve tahminlerini istek izleme sistemindeki tüm önceki isteklere dayanarak üretir. Böylece sistem, insan bilgi dağarcığına sığmayacak kadar bilgi ile eğitilmiş olur.
- Otomatik sistem karar verirken diğer bireylerden etkilenmez veya herhangi bir baskı hissetmez. Bilgiye dayalı, tarafsız tahminler sunar.
- Tahminler tekrarlanabilirdir. Makine yorulmaz ve her zaman tutarlı çıktılar sağlar.
- Otomatik sistem, insan yargısına dayalı grup kararlarından çok daha hızlı sürede tahmin çıktıları elde edilmesini sağlar. Böylece hem zamandan hem insan kaynağından tasarruf sağlanır.

Bu tezde yazılım projelerinde efor tahmininin otomatik bir sistemle gerçekleştirilebileceği derin öğrenme tabanlı bir yöntem sunulmuştur. Böylece çok daha büyük verilerle eğitilen otomatik bir sistemle, çok daha tutarlı, tekrarlanabilir ve hızlı tahmin sonuçları elde edilebilecektir.

Tablo 2.1'de, 2007'den 2018'e kadar 10 yıl boyunca literatürde çalışılan makine öğrenme tekniklerinin bazı örnekleri listelenmiştir. Bu tablo, son yıllarda yapılan yazılım efor tahmini çalışmalarının, yeni bir makine öğrenmesi tekniği önermek yerine, geliştirme tekniklerinin keşfi üzerinde olduğunu göstermektedir (Arbain ve diğ., 2019).

Tablo 2-1 Yazılım efor tahmini için 10 yılda uygulanan bazı makine öğrenmesi teknikleri

Yıl	**Teknikler**	**Veri Seti**	**Değerlendirme Metodu**
2007,2015	Nöro Bulanık	NASA & COCOMO	MMRE
2008	PSO	NASA	MMRE
2008	Regresyon Teknikleri	COCOMO	MMRE, PRED
2010,2013,2015	Genetik Algoritma	-Türk & Endüstri Veri Seti-COCOMO & NASA	
2011,2014	YSA	COCOMO & NASA, PROMISE deposu	
2011,2018	Bulanık Mantık	NASA	MMRE, PRED
2013	PSO-YSA- COCOMOII	COCOMO I & NASA93	MRE, PRED(25%)
2014	Radyal Tabanlı Fonksiyon NN	COCOMO & NASA	MSE
2014	PSO ile bayes	NASA93	MMRE
2014	YSA ile Temel Bileşenler Analizi	COCOMO 81	
2014	Benzetimli Tavlama	NASA	
2014	Hibrit COCOMO ve Fonksiyon Noktası	NASA	MRE
2015	Karınca Kolonisi Optimizasyonu	Etkileşimli Sesli Yanıt Yazılım projesi veri seti	MMRE
2016,2017,2018	Guguk kuşu, Hibrit Guguk kuşu Opt.	NASA60, NASA63, NASA93,MAXWELL, KEMERER, MIYAZAKI	MRE, MMRE
2016	COCOMO Öğretme Öğrenme Esaslı Opt.	NASA	MMRE, PRED(30%)

2016	Diferansiyel Değerlendirme Metodu	NASA	MMRE, MRE
2016	Hibrit PSO-ANFIS	NASA	MSE, MMRE, PRED
2016	Yarasa Algoritması	NASA93	MMRE
2016	COCOMO-Genetik Algoritma	NASA	MMRE
2017	YSA-Yusufçuk Algoritması	NASA	MSE, MMRE, MAE

2.4.2. Literatürde Makine Öğrenmesiyle Yazılım Efor Tahmini

Literatürde yazılım projelerinde efor tahmini için, uygulanan proje yönetim metodolojilerine, eforun temsil edildiği göstergelere, kullanılan veri setlerine, performans değerlendirme metriklerine, uygulama platformuna ve uygulanan yöntemlere göre farklılıklar gösteren, makine öğrenmesi algoritmalarıyla yapılmış bir çok çalışma bulunmaktadır. Bu çalışmalara bazı örnekler aşağıda açıklanacaktır.

Khuat ve Le tarafından 2017'de yapılan bir çalışmada öğretme-öğrenme temelli optimizasyon yöntemi ile bal arılarının beslenme davranışından esinlenen Yapay Arı Kolonisi yönteminin olumlu yönleri birleştirilerek yeni bir yaklaşım önerilmiştir. Önerilen yöntemin performansı, proje yönetiminde karmaşık ve önemli bir adım olan yazılım eforu tahmin problemi üzerinde değerlendirilmiştir. Veri seti olarak COCOMO II Nasa yazılım proje veri seti kullanılmıştır. Elde edilen sonuçlar, önerilen yeni yöntemin orijinal COCOMO II modeline kıyasla daha iyi tahmin yetenekleri sağladığını göstermiştir. Bunun yanı sıra Yapay Arı Kolonisinin yeni bir meta-buluşsal optimizasyon algoritması olmasına rağmen, yakınsama hızı ve çözüm kalitesi bakımından hala yetersiz olduğu görülmüştür.

Nwaiwu ve Oluwadere tarafından 2016'da yapılan bir çalışmada, 93 COCOMO Nasa projesinin verileri kullanılarak, bulanık tabanlı modellerin tahmin doğruluğunu etkileyen çeşitli koşullarını analiz etmek için, iki bulanık model geliştirilmiştir. Bunların biri kontrol grubudur. Diğeri ise model yapısının, dilsel değişkenlerin ve girdi-çıktı parametrelerinin koşullarına göre değişen birçok farklı deneysel grup olarak açıklanabilir. Bu gruplar üzerine modeller kurularak karşılaştırma yapılmıştır. Performans değerlendirmesi için MMRE (Göreceli Hatanın Ortalama Büyüklüğü) ve PRED (Tahmin) değerleri kullanılmıştır. Bulanık modelde dilsel değişkenlerin ve girdi değişkenlerinin sayısındaki artışın, performans üzerinde olumlu sonuçlar doğurduğu

gözlemlenmiştir. Bununla beraber bulanık tabanlı bir yazılım efor tahmin modelinin, geleneksel modellere kıyasla daha iyi bir performans gösterdiği görülmüştür.

Laqrichi ve diğ tarafından 2015'te yapılan çalışmada, veri setindeki belirsizlikler dikkate alınarak gerçekçi efor tahminleri sağlayan bir yazılım efor tahmin yöntemi önerilmiştir. Çalışma, Uluslararası Yazılım Karşılaştırma Standartları Grubu 2011 veri seti üzerine üç aşamada gerçekleştirilmiştir. Veri seti hazırlığında Bootstrap çalışmaları da yapılmıştır. 5 ayrı YSA (Yapay Sinir Ağları) modeliyle efor tahmini yapılarak MMRE değerleri hesaplanmıştır. Model üzerine bootstrap çalışması yapıldığında daha güvenilir tahminler elde edildiği gözlemlenmiştir.

Agrawal ve Shrivastava tarafından 2016'da yapılan bir çalışmada, uyarlamalı ağ tabanlı bulanık çıkarım sistemi kullanılmıştır. Ağ, geri besleme ve en küçük kareler algoritması kombinasyonundan oluşan hibrit bir öğrenme algoritması yaklaşımı kullanılarak eğitilmiştir. Yazılım geliştirme eforu, 41 projeden oluşan Lopez Martin veri setini kullanarak üç üyelik fonksiyonu(Gauss, Üçgen ve Trapezoidan) için hesaplanmıştır. Bu çalışmada, üç farklı üyelik fonksiyon modeli kullanan nöro bulanık modeller, 5 farklı performans değerlendirme metriğine göre karşılaştırılmıştır. Karşılaştırma sonucunda, Trapezoidal üyelik fonksiyonunu kullanan nöro bulanık modelin diğer tüm modellerden beş metriğin hepsi için daha iyi sonuçlar verdiği görülmüştür.

Hernandez ve diğ. tarafından 2015'te yapılan çalışmada, geri besleme yönteminin avantaj ve dezavantajları gözlemlenmiştir. Geri besleme yöntemi, çok katmanlı sinir ağlarının eğitilmesiyle oldukça avantajlı sonuçlar vermesine karşın, olasılıklı tahmin durumları için dezavantajlara sahiptir. Bu çalışmada, olasılıksal sinir ağlarını eğitmek için yeni bir ölçeklenebilir yöntem sunulmaktadır. Olasılıksal geri besleme yöntemi, klasik geri besleme yönteminde olduğu gibi ağ üzerinden olasılıkların ileri yayılımını hesapladıktan sonra geriye doğru hesaplamasını yaparak çalışmaktadır. 10 gerçek veri seti üzerine yapılan bir dizi deney, olasılıksal geri besleme tekniğinin, diğer tekniklerden önemli ölçüde daha hızlı sonuç verdiğini ve böylece rekabetçi bir tahmin yeteneği olduğunu göstermiştir.

Rijwani ve Jain'in 2016'da yaptığı bir çalışmada yazılım geliştirme eforunu tahmin etmek için, geri besleme yöntemi ile eğitim verilen çok katmanlı ileri beslemeli sinir ağları kullanılarak

geliştirilmiş YSA tabanlı bir model önerilmektedir. Önerilen bu yöntem, yazılım efor tahmininde en sık kullanılan algoritmalardan biri olan COCOMO yöntemi karşılaştırılmıştır. Performans ölçüm metrikleri olarak MSE (Ortalama Hata Karesi) ve MMRE kullanılmıştır. Ağın test edilmesinde ve eğitilmesinde COCOMO veri seti kullanılmıştır. Uygulamadaki Sinir Ağı Modeli, MATLAB yazılımında bulunan Sinir Ağı toolbox22 kullanılarak oluşturulmuştur. Çalışmanın çıktılarına göre önerilen model, COCOMO modeline göre daha iyi sonuçlar vermiştir.

Ebrahimpour ve diğ. tarafından 2016'da yapılan bir çalışmada çok katmanlı algılayıcı karınca kolonisi optimizasyonu adı verilen, çok katmanlı algılayıcı yapay sinir ağı ve karınca koloni optimizasyonu algoritmasının yeni bir hibrit modeli sunulmuştur. Veri seti olarak COCOMO Nasa veri seti içerisinde geçerliliği kabul edilmiş 25 projenin verisi kullanılmıştır. Sonuçlar, % 80'den fazla örnekte yüksek doğrulukta olan bu yeni yaklaşımın COCOMO modelinden daha iyi olduğunu, yanı sıra MMRE değerinin COCOMO modelindekinden daha düşük olduğunu göstermektedir.

Nassif ve diğ. tarafından 2020 yılında yapılan bir çalışmada, yazılım efor tahmini üzerine doğrusal olmayan regresyon analizine dayalı yeni bir model önerilmiştir. Yazılım efor göstergesi olarak kullanım durumu puanları ele alınmıştır. 146 projeden oluşan veri seti, kullanım durumu puanı 100'ün altında olanlar, 100-300 arası olanlar, 300'ün üstü olanlar ve tüm veri seti olmak üzere 4 ayrı modelde test edilmiş ve 7 ayrı performans metriğine göre karşılaştırılmıştır. Önerilen doğrusal olmayan regresyon modelinin, orijinal kullanım durumu puanı modelinden daha iyi performans gösterdiği görülmüştür.

Shahul ve diğ. tarafından 2016'da yapılan bir çalışmada, makine öğrenmesi tekniklerine dayalı yazılım maliyet/efor tahmini için veri ön işleme yöntemleri incelenmiştir. Veriler, doğrusal regresyon yönteminin iyi tahminlerde bulunacağı şekilde düzeltilmiştir. Çalışmada bir yazılım maliyet tahmin veri seti olarak kullanılan Kitchenham veri seti kullanılmıştır. Bu veri seti, Rapid Miner aracılığıyla bir ön işleme sürecinden geçirilmiştir. Ön işleme teknikleri ile makine öğrenmesi yöntemlerinin tahmin doğrulukları arasındaki etkileşim incelenmiştir. Veri ön işleme adımlarının, yazılım maliyet tahmini için kullanılan bir makine öğrenme yönteminin etkinliğini önemli ölçüde değiştirebildiği gözlemlenmiştir.

Kangwantrakool ve diğ. tarafından 2020'de yapılan bir çalışmada, yazılım geliştirme projelerinde efor tahmini için metin ön işleme süreçlerinden faydalanılarak bazı derin öğrenme teknikleri önerilmiştir. Çok Katmanlı Algılayıcı ile ortalama sözcük-vektör, Destek Vektör Regresyonu ile ortalama sözcük-vektör, Geçmeli Tekrarlayan Birim dizi modeli ve Uzun Kısa Dönem Hafızası dizi modeli olmak üzere dört mimari; adam-ay farkı açısından karşılaştırılmıştır. Birincisi ham metin, ikincisi bir yazılım projesinin özelliklerini açıklayan yapılandırılmış veri tablosu olmak üzere, Isem ve Isbsg adında iki ayrı yazılım projesi veri seti kullanılarak değerlendirilmiştir. Metin verilerine bir dizi metin ön işleme adımı uygulanmıştır. Uzun Kısa Dönem Hafızası dizi modeli, iki veri seti için en düşük ve ikinci en düşük MAE (Ortalama Mutlak Hata) oranlarına ulaşmıştır.

Malhotra ve diğ. tarafından 2011'de yapılan bir çalışmada yazılım efor tahmini için Karar Ağacı, Doğrusal Regresyon, Yapay Sinir Ağı, Destek Vektör Makinesi ve Torbalama yöntemleri uygulanarak MMRE başta olmak üzere 6 ayrı performans ölçümüyle karşılaştırılmıştır. 499 projeden elde edilen bir veri seti kullanılmıştır. 19 öznitelikten oluşan veri seti Korelasyon Temelli Öznitelik Alt Seçimi (CFS) yöntemi ile 10 özniteliğe düşürülmüştür. Sonuçlar, literatürdeki 24 model ile karşılaştırılmıştır ve çalışmada önerilen karar ağacı modelinin efor tahmininde en iyi MMRE değerlerini verdiğini göstermiştir.

Feridana ve diğ. tarafından 2011'de yapılan bir çalışmada çevik projelerde kullanıcı hikaye puanı tahmini üzerine Dağıtılmış Ekstrem Programlama (DXP) tekniği önerilmiştir. Bir imalat şirketi için muhasebe ve ürün dağıtım sisteminin geliştirildiği bir proje üzerinde uygulanmıştır. Bu tahmin tekniği, yazılımın karmaşıklığını ve yazılımı oluşturmak için gereken adam-gün eforunu tahmin edebilmektedir. Bunun için öncelikle 13 teknik faktör, 9 deneyim faktörü karmaşıklığına göre ağırlıklandırılmıştır. 687 düzeltilmemiş kullanıcı hikaye puanından oluşan 406 kullanıcı hikayesi verisi kullanılmıştır. Sonucunda projenin en ideal kaç çalışma gününde ve kaç kişiyle tamamlanacağına dair alternatifler sunulmuştur.

Scott ve diğ tarafından 2018'de yapılan bir çalışmada çevik yazılım geliştirme projelerinde efor tahmini üzerine Destek Vektör Makinasına dayanan bir yaklaşım önerilmiştir. 8 açık kaynaklı projenin istek raporlarından oluşan bir veri seti kullanılmıştır. Veri seti üzerine veri madenciliği işlemleri uygulanmıştır. Yalnızca geliştirici özniteliklerinin, yalnızca metinsel özniteliklerin ve ikisinin kombinasyonundan oluşan özniteliklerin kullanıldığı modeller karşılaştırılmıştır.

Performans değerlendirmesi için Ortalama Mutlak Hata ve Standart Doğruluk metrikleri kullanılmıştır ve yalnızca geliştirici özniteliklerinin kullanıldığı modelin en iyi sonuçları verdiği gözlemlenmiştir.

Porru ve diğ. tarafından 2016'da yapılan bir çalışmada çevik projelerde hikaye puanlarını tahmin etmek için Destek Vektör Makinesi, K-En Yakın Komşu, Karar Ağacı ve Naive Bayes olmak üzere 4 farklı makine öğrenimi sınıflandırıcısı karşılaştırılmıştır. 8 açık kaynaklı projenin verileri kullanılmıştır. Performans metriği olarak MMRE değerleri baz alınmıştır. Karşılaştırma sonucunda Destek Vektör Makinesi yönteminin en düşük MMRE'yi verdiği gözlemlenmiştir. Sistem kurulduktan sonra yeni bir isteğin tahmin süresinin 8 ila 15 saniye arasında olduğu görülmüştür.

2.5.DERİN ÖĞRENME VE YAPAY SİNİR AĞLARI

2.5.1. Derin Öğrenme ve Yapay Sinir Ağları

Derin öğrenme, yakın zamanda ele alınan yapay zeka alanındaki birçok probleme umut verici bir çözüm olarak dikkat çekmektedir (Kim ve diğ, 2017). Makine öğrenmesinin yeni bir alanı olan Derin Öğrenme, son yıllarda geliştirilen yapay zekâ uygulamalarının başarısını oldukça üst düzeylere çıkarmıştır. Bilgisayarların hesaplama gücündeki büyük ilerlemelerin de etkisiyle, karmaşık ve yüksek miktarda veriyi işleyebilecek donanımlara ihtiyaç duyan derin öğrenme algoritmaları sıkça kullanılmaya başlanmıştır.

Derin sinir ağı algoritmaları, insan beyninin öğrenme sonucu bilgi elde etme becerisinden esinlenerek geliştirilen yapay sinir ağlarının çok katmanlı halidir. En ilkel yapay sinir ağı olarak bilinen tek katmanlı algılayıcının, doğrusal olmayan problemlerin çözümünde yeterli olmaması nedeniyle çok katmanlı algılayıcılar geliştirilmiştir. Derin yapay sinir ağları, birden fazla gizli katmana sahip yapay sinir ağlarıdır. Burada derinlik gizli katman sayısı artırılarak sağlanır.

Derin öğrenme, birden fazla gizli katman içeren bir yapay sinir ağları yaklaşımıdır. Bu anlamda derin sinir ağları(DSA), YSA'nın yapısal açıdan daha karmaşık hali olarak düşünülebilir. Derin öğrenme, tek bir katmandaki özelliği daha üst katmandaki bir özelliğe dönüştüren basit, fakat doğrusal olmayan işlem birimlerini kapsar. Dolayısıyla, derin öğrenme yaklaşımı, çok katmanlı öznitelikleri öğrenebilen bir metottur. Verilerin çoklu seviyelerde gösterimini öğrenmek için

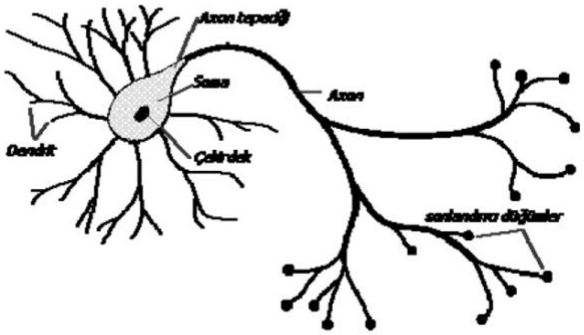
çoklu işleme katmanlarından oluşan hesaplama modellerine izin verir. Örneğin, bir görüntü işleme projesinde görüntüleri tanımak için derin öğrenme ağları, ilk katmandaki ham piksel girdilerinden renk bilgisini öğrenebilir ve bu renk bilgisini bir sonraki katmandaki nesnelerin kenarlarına dönüştürebilir (Zhang ve diğ, 2017) ya da bir başka görüntü işleme projesinde derin öğrenme ağları, ilk katmanda kenarları oluşturabilir, ikinci katmanda bu kenarlardan yola çıkarak ilgili objenin diğer parçalarını tespit edebilir, daha üst katmanlarda ise bu parçaları birleştirerek objenin şekillerini ayırt edebilir. Bu örneklere ek olarak derin öğrenme, konuşma tanıma, duygu analizi, görsel nesne tanıma ve nesne algılama gibi birçok alanda da son teknolojiyi önemli ölçüde geliştirmiştir.

Derin Sinir Ağları, Derin İnanç Ağları(DBN), Evrişimsel Sinir Ağları(CNN), Tekrarlayan Sinir Ağları(RNN) gibi farklı derin öğrenme mimarileri mevcuttur. Derin evrişimsel ağları, görüntü, video, konuşma ve ses işlemede büyük başarılar gösterirken; tekrarlayan ağlar, metin ve konuşma gibi ardışık veriler üzerine ışık tutmaktadır (Lecun ve diğ, 2015). Projenin amacına ve veri setine uygun bir derin ağı mimarisi seçmek, projenin başarısı doğrudan etkilemektedir.

Sonuç olarak derin sinir ağlarının en temelde birden fazla gizli katmanı olan yapay sinir ağları olarak tanımlandığı düşünülürse derin yapay sinir ağlarının yapısını anlayabilmek için yapay sinir ağlarını anlamak gerekmektedir.

2.5.2. Yapay Sinir Ağları ve Biyolojik Sinir Ağları

Yapay sinir ağlarının kavramsal temeli, insan beyninin sinirsel mimarisi üzerinde yapılan çeşitli araştırmalara dayanmaktadır (Gritta, Davalos ve Wang, 2002). İnsan sinir hücresi çekirdek, gövde ve iki uzantıdan oluşmaktadır (Şekil 2.2). Uzantıların kısa ve dallanmış olanı dentrittir. Dentritin görevi girdi bilgilerini almaktır. Tek ve uzun olan uzantıya ise akson adı verilir. Aksonun görevi, çıktı bilgilerini diğer sinir hücrelerine taşımaktır. Akson ve dentritin birleştiği yere sinaps denir ve yaklaşık olarak 10^{14} sinaps olduğu tahmin edilmektedir (Pandya ve Macy, 1996).



Şekil 2-2 Temel insan sinir hücresi(Anderson ve diğ., 1992)

YSA algoritmaları insan beynindeki öğrenme işleyişinden hareketle geliştirilmiştir. Biyolojik sinir sisteminde bulunan nöronların birbirleri ile ilişki kurması gibi YSA sistemlerinde de nöron şeklinde tanımlanan yapılar birbirleri ile bağlantılı olacak şekilde modellenmişlerdir. Algoritmanın bu şekilde öğrenme, hafızaya alma ve veriler arasındaki ilişkiyi ortaya çıkarma kapasitesine sahip olacağı düşünülmüştür.

İnsan beyni, birbirleriyle bağlantılı işlem elemanlarından(nöronlardan) oluşmaktadır. YSA'larda nöron, biyolojik nöronlardan daha az bağlantıya sahip olma eğilimindedir. YSA'larda da insan beyninde olduğu gibi öğrenme süreci sonucunda bilgi elde edinilir. Bu bilgileri depolamak için sinir hücreleri arasında sinaptik ağırlık olarak adlandırılan nöronlar arası bağlar kullanılmaktadır. Her bir nöron, bir diğer nöronun çıktısını veya dışarıdan gelen bir uyarıyı girdi sinyali olarak alır, aktivasyon fonksiyonu ile işler ve bir diğer nörona veya dışarıya verilmek üzere çıktı sinyali üretir (Taylor, 2005). Sinyallerin iletilmesi ve işlenebilmesi için insan beyninde eşik değerinin aşılması gerekmektedir; aksi takdirde iletim gerçekleşmez. Bu yapı, yapay sinir ağında da eşik değeri verilerek sağlanmaktadır. Yapay ve Biyolojik Sinir Hücresi arasındaki ilişki Tablo 2.2'de (Efe ve Kaynak, 2000) gösterilmektedir.

Tablo 2-2 Biyolojik sinir hücresi ile yapay sinir hücresi arasındaki ilişki

Biyolojik Sinir Hücresi	**Yapay Sinir Hücresi**
Sinir Hücresi	Düğüm, Nöron, İşlem elemanı, Algılayıcı
Sinaps	Ağırlıklar
Dendrit	Toplama fonksiyonu (Giriş)
Hücre Gövdesi	Aksiyon(Etkinlik) fonksiyonu
Akson	Sinir Çıkışı (Çıkış)

Yapay sinir ağları, birbiriyle bağlantılı veya birbirine bağlı birimleri temsil edebildiğinden, bir yazılım ürününün üretimindeki çeşitli faaliyetleri temsil etmek için de yetenekli bir araç olmaktadır. Sinir ağında, her birim (bir nöron olarak anılır ve şebeke düğümü tarafından temsil edilir) bir etkinliği temsil eder. Her etkinliğin girdi ve çıktıları vardır. Ağın her bir birimi, girdilerinin hesaplamasını gerçekleştiren, ağırlıklandırılmış toplam hesaplayan bir sürece girer. Birim toplam eşik değerini aşarsa, ağ bir çıktı üretir. Sonuç olarak çıktı, ağ tarafından nihai bir çıktı değeri üretilene kadar, ağdaki diğer ilgili birimlere girdi olur (Rumelhart ve diğ., 1986).

Yapay Sinir Ağları, regresyon, zaman serileri tahmini ve sınıflandırma gibi çeşitli sorunlara başarıyla uygulanmıştır. Bir Yapay sinir ağı projesinde performansı etkileyen temel bazı hiperparametreler; gizli katmanların sayısı, gizli katmanların nöron sayısı, eğitim devirlerinin sayısı, öğrenme oranı ve momentum olmaktadır. Verilen bir problemde ağın performansını artırmak için bu parametreler dikkatle seçilmelidir (Braga ve diğ, 2007). Bu konunun detaylarına 3.3. Derin Sinir Ağı Hiperparametreleri başlığı altında yer verilmiştir.

2.5.3. Yapay Sinir Ağlarının Yapılarına Göre Sınıflandırılması

Yapay sinir ağları, geçmiş projelerden elde edilen verilerle ağa "eğitim" vererek geliştirilir. İlgili veriler ağa ulaştırılır ve ağ, verideki dokuları tanımlayarak "öğrenmek" için ileri ve geri algoritmalar kullanır. Geçmiş projelerle ilgili veriler örneğin yazılımcı deneyimi, bir projeyi tamamlamak için gereken eforun miktarı gibi bilgiler içerebilir (Kosinar ve Strba, 2013).

Bir yapay sinir ağındaki çıktıları üretmenin birçok yolu vardır. En yaygın yöntemlerde biri, geriye dönük aktivasyon grafiklerinin kullanıldığı yöntemdir. Önceki nöronlara dönülen bir süreç söz konusudur. Bunlara geri besleme teknikleri denir. Bu yöntemlerden bir diğeri ise tahmin için ileriye dönük süreçleri içermektedir (Rumelhart ve diğ., 1986).

YSA yaklaşımları, yapılarına göre dört tür halinde sınıflandırılabilir. Bunlar ileri beslemeli, geri beslemeli, radyal ve kendi kendini organize eden sinir ağlarıdır. Bu mimariler arasında geri beslemeli sinir ağları en popüler yapay sinir ağı yöntemlerinden biridir (Zhang ve diğ, 2017).

2.5.3.1. Geri Beslemeli Sinir Ağları:

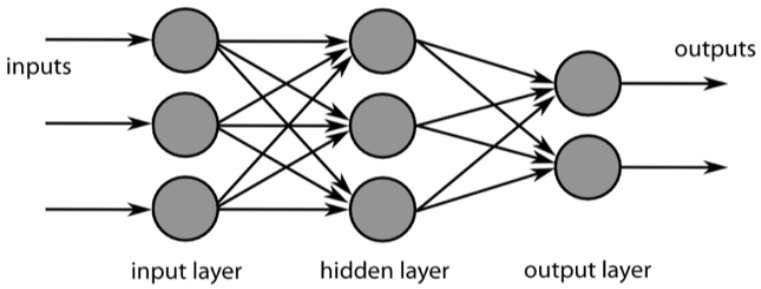
Geri besleme, bir sinir ağının eğitilmesi ya da çıktılarının üretilmesi için kullanılan en yaygın yöntemdir. Bu yöntem, ağ ağırlıkları bakımından kaybın derecesini hesaplamak için iki aşamada çalışır. Birinci aşamada girdiler, fonksiyon çıktısını ve dolayısıyla mevcut parametrelerle ilişkili kaybı hesaplamak için ağırlıklar ağda ileriye doğru yayılır (Rumelhart ve diğ., 1986). İkinci aşamada hesaplanan hata sinyalleri, ağırlıkların ayarlandığı ağ üzerinden geriye doğru yayılır. Çıktının hesaplanması, katmanlara göre ileri yönde yapılmaktadır. Bir katmanın çıktısı bir sonraki katmanın girdisidir (Kumar ve diğ, 2002).

Her katmandaki gösterim, önceki katmandaki gösterimden yola çıkarak hesaplanırken parametre değişimleri, geri besleme algoritması kullanılarak gerçekleştirilir ve karmaşık veri kümelerindeki yapı, bu süreçte keşfedilir (Lecun ve diğ, 2015).

Geri beslemeli sinir ağı, ağ çıktıları ile eğitim kümesindeki test verileri arasındaki hataları en aza indirgeyen metotlardan biridir. Geri beslemeli sinir ağı kendini organize etme ve kendi kendine öğrenme kabiliyetleri ile karakterizedir (Zhang ve diğ, 2017).

2.5.3.2. İleri Beslemeli Sinir Ağları:

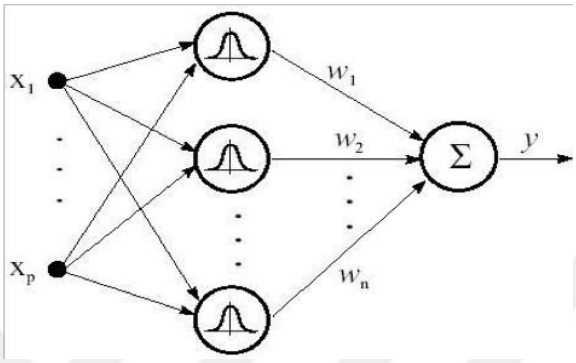
Bir ileri beslemeli sinir ağı girdi katmanı ile başlar. Girdi katmanı, gizli bir katmana bağlanır. Bu gizli katman, eğer varsa başka bir gizli katmana bağlanır, yoksa çıktı katmanına bağlanır. Sinir ağlarında en az bir gizli katman vardır. Şekil 2.3'te de görüldüğü gibi süreç ileri doğru işler.



Şekil 2-3 İleri Beslemeli Sinir Ağları

2.5.3.3. Radyal Temel Fonksiyonu Sinir Ağları:

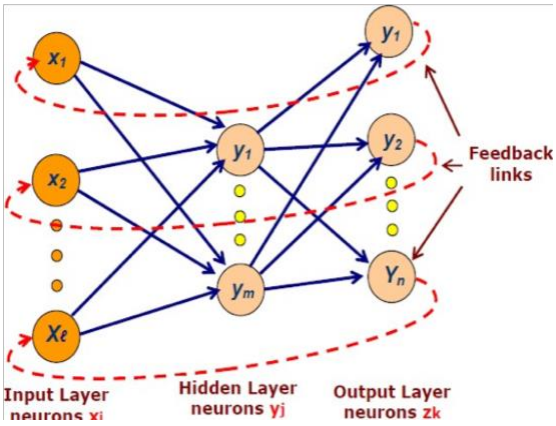
Radyal temel fonksiyonu sinir ağları, denetimli öğrenme algoritması kullanılarak eğitilmiş, ileri beslemeli ağlardır. Bu ağlarda genellikle aktivasyon fonksiyonu, temel fonksiyonlar olarak adlandırılan bir fonksiyon sınıfından seçilen tek bir gizli katman ile yapılandırılır. Birçok yönden geri beslemeye benzer olsa da, radyal temel fonksiyon ağlarının farklı avantajları vardır. Genellikle geri beslemeli ağlardan çok daha hızlı çalışırlar (Şekil 2.4).



Şekil 2-4 Radyal Temel Fonksiyonu Sinir Ağları

2.5.3.4. Tekrarlayan Sinir Ağları:

Tekrarlayan bir sinir ağının temel özelliği, ağın en az bir geri besleme bağlantısı içermesidir. Kendi kendine geri bildirim bağlantıları olan nöronlar da yer alabilir. Bu da bir nöronun çıktısının, girdi olarak geri dönüşünün olduğu anlamına gelir. Şekil 2.5'de bu döngü gösterilmiştir (Panchal and Panchal, 2014).



Şekil 2-5 Tekrarlayan Sinir Ağları

2.6.KARAR AĞAÇLARI

Karar Ağaçları, veri kümesinin belirli bir kritere göre bir ağaç yapısı şeklinde yinelemeli bölünerek kümelendiği hiyerarşik bir makine öğrenmesi yöntemidir.

Bu bölümde öncelikle karar ağaçlarının temel yapısı anlatılacaktır. Karar ağacı bölünmeleri, hedef değişkenin veri tipine göre regresyon veya sınıflandırma ağaçları oluşturacak şekilde gerçekleşir. Bu çalışmada regresyon modeli uygulandığı için ilerleyen başlıklarda regresyon ağaçları üzerinde durulacaktır. Daha sonra karar ağaçlarında performansı artırmak amacıyla başvurulan model birleştirme yöntemlerinden bahsedilerek, çalışmada kullanılan gradyan güçlendirme yöntemi ve karar ağacı regresyonlarında kullanılma şekli açıklanacaktır.

2.6.1. Karar Ağaçlarının Yapısı ve Oluşturulması:

Karar ağaçları, kök düğüm, iç düğümler ve yapraklardan oluşur. Kök düğüme gelen dal yoktur; sıfır veya daha fazla giden dal vardır. Kök düğüm, veri kümesindeki tüm verileri(gözlem, örnek) içerir. İç düğümlerin her birinin bir gelen dalı ve iki veya daha fazla giden dalı vardır. İç düğümler ve kök düğüm, veri kümesindeki değerleri özelliklerine ayırmak için kullanılan kuralları içerir. Yaprakların her birinin bir gelen dalı vardır ve giden dalı yoktur. Yapraklar, hedef değişkenin tahminlerini içerir (Ferreira, 2016).

Karar ağaçları, açıklayıcı değişkenlere dayalı her bir düğümün ard arda ikiye bölünmesiyle oluşturulur. Her bir açıklayıcı değişken, biri sol ve diğeri sağda olmak üzere iki alt düğümden birine gönderilir. Hedef değişken sürekli olduğunda regresyon ağaçları, kategorik olduğunda sınıflandırma ağaçları söz konusudur.

Sürekli bir açıklayıcı değişken için tek bir bölünme noktası belirlenir. Açıklayıcı değişken değeri bu bölünme noktasından küçük olan gözlemler sola, diğer gözlemler ise sağa bölünür. Kategorik bir açıklayıcı değişken için ise bölünme, kategorilerin bir alt grubunu sola, diğerlerini ise sağa ayırır. İkili bölünmelerle her seferinde daha küçük alt gruplar oluşturulur.

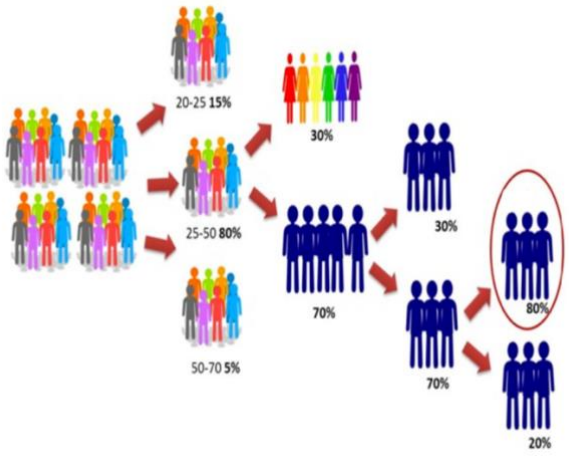
Bölünme süreci bir durdurma kriterine ulaşana kadar kendi çıktısına tekrarlı olarak uygulanır. Durdurma kriterine örnek olarak, tüm düğümlerin belirlenmiş bir sayıdan daha az sayıda gözlem içermesi durumunda bölünmenin durdurulması verilebilir.

Durdurma kriterine ulaşıldıktan sonra aşırı uyumu önlemek için ağaç, tespit edilen en zayıf bağlantıları kırılarak budanır. Tahmin değerleri, budama sonrasında ağaçta kalan bölünmemiş düğümlerdeki (yapraklardaki) gözlemlerden elde edilir. Tahmin değerleri, regresyon problemlerinde hedef değişkenin ortalaması alınarak, sınıflandırma problemlerinde ise sınıf üyeliklerinin oranları hesaplanarak elde edilir.

Bu çalışmada hedef değişken, sayısal verilerden oluşan hikaye puanlarıdır. Bu nedenle bölünmeler regresyon ağacı yöntemine göre olacaktır. Regresyon ağacı modelleri bir sonraki bölümde anlatılacaktır.

M yapraklı ikili bir ağaçta M tahmin sayısını gösterirken, M-1 kural sayısını göstermektedir (Işıkhan, 2014). Örneğin kök düğümün 2'ye bölündüğünü, sonra bu iki iç düğümün de 2'ye bölündüğünü ve bölünmenin durduğunu düşünelim. 4 tane yaprak oluşacaktır. Bu durumda kök düğüm ve altındaki iki düğüm, 3 adet kuralı içerecektir.

Şekil 2.6'da (Dixit, 2017) örnek bir karar ağacı gösterilmektedir. Kök düğüm, yatırımla ilgilenen tüm nüfustan oluşmaktadır.



Şekil 2-6 Karar Ağacı Örneği

Kök düğüm, yaş grubuna göre 3 iç düğüme bölünmektedir. Yatırımla ilgilenen nüfusun %15'i 20-25 yaş arasında, %80'i 25-50, %5'i ise 50-70 yaş arasında yer almaktadır. Bu gruplar arasında 25-50 yaş grubunun %80 nüfusla büyük çoğunluğu içermesinden dolayı bu iç düğüm ile ilerlenmiştir.

Bir sonraki aşamada 25-50 yaş grubunu içeren iç düğüm, Kadın ve Erkek olmak üzere iki alt gruba ayrılmıştır. Böylece bu gruptaki nüfusun %30'unun kadın, %70'inin ise erkek olduğu görülmüştür. 25-50 yaş grubu erkek nüfusundaki yoğunluk nedeniyle %70'lik iç düğüm ile ilerlenmiştir.

Daha sonraki aşamada, belirlenen erkek nüfusu, medeni durumuna göre Evli ya da Bekar olmak üzere ikiye bölünmüştür. Böylece yatırımla ilgilenen 25-50 yaş grubu bekar erkeklerin oranı %30 iken evli erkeklerin oranının %70 olduğu gözlemlenmiştir ve evli erkek oranının yoğunluğundan dolayı %70'lik iç düğüm ile ilerlenmiştir.

Son aşamada, belirlenen evli erkek nüfusu, ne kadarının çocuk sahibi olduğuna göre ikiye bölünmüştür. %80'i çocuk sahibi iken, %20'sinin çocuk sahibi olmadığı gözlemlenmiştir. Böylece piyasada en fazla yatırımı, 25-50 yaş arası evli ve çocuk sahibi olan erkeklerin yaptığı görülmüştür.

Karar ağacı algoritmaları, veri kümesinin dallar aracılığıyla hangi yollarla bölümlere ayrılacağını belirlemektedir. Dolayısıyla belirli bir veri kümesinden oluşturulabilecek birçok farklı karar ağacı vardır. Bu ağaçlar, farklı doğruluk oranlarına sahip olmakla birlikte, aralarında en uygun olan ağacı aramak, pek hesaplanabilir değildir. Bu nedenle uygun bir süre içerisinde yetersiz bir karar ağacı da olsa makul bir doğruluk oranına ulaşmak amacıyla bir dizi etkili algoritma geliştirilmiştir.,

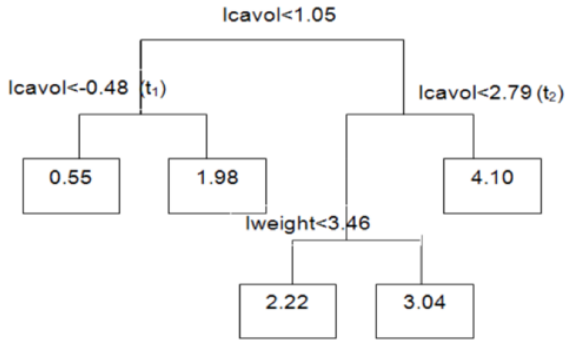
Karar ağaçlarını oluşturmak için en yaygın kullanılan algoritmalar şöyledir; ID3 algoritması, C4.5 ve C5.0, Otomatik Etkileşim Tespiti (AID) Sınıflandırma ve Regresyon Ağacı (CART), CHAID ve QUEST

2.6.1. Regresyon Ağaçları

Regresyon ağacı, sürekli bir "y" hedef değişkeninin, sürekli, ordinal ve nominal değişkenlerin rastgele bir karışımı olabilen "$x = x_1 + x_2 + \cdots + x_n$" açıklayıcı değişken vektörü tarafından açıklandığı bir yöntemdir. Regresyon ağacı, açıklayıcı değişkenler üzerinde karar kuralları seti oluşturur.

Regresyon ağacı modelleri, en homojen yanıtlara sahip bölgeleri belirlemek için, açıklayıcı değişken uzayını bir dizi kural kullanarak dikdörtgenlere ayırır. Her bir bölge için, ilgili bölgedeki gözlemlerin ortalaması sabit alınır.

Regresyon ağaçlarının gösterimine örnek olması bakımından Stamey ve diğ. tarafından 1989'da prostat kanseri üzerine yapılan bir çalışma ele alınmıştır. Bu çalışmada elde edilen veriler için oluşturulan regresyon ağacı Şekil 2.7'de(Stamey ve diğ, 1989) gösterilmektedir.



Şekil 2-7 Regresyon Ağacı Örneği

Modelde log kanser hacmi, lcavol ve log prostat ağırlığı, lweight olmak üzere iki açıklayıcı değişken bulunmaktadır. Hedef değişken ise prostata özgü antijen seviyesi olan lpsa'dır. Düğümler, eşitsizliği sağlayan gözlemler sola, sağlamayan gözlemler sağa olmak üzere ikili bölünmelerle oluşturulmuştur. Düğümlerde gösterilen değerler, her düğümü oluşturan gözlemlerin ortalama değerleridir. Son düğümler ise tahmin değerlerini göstermektedir ve ilgili bölgedeki hedef değişkenlerin ortalamalarından oluşur. Tahmin sürecinde sadece ağaçtaki sabitlere bakıldığı için hesaplama hızlıdır.

2.6.2. Model Birleştirme Yöntemleri

Bir karar ağacının performansını artırmak için çeşitli yöntemler bulunmaktadır. Bunlardan bir tanesi model birleştirmedir. Model birleştirme (ensemble) yöntemleri makina öğrenmesinde, tek bir model bileşeninden elde edilebilen performanstan daha iyi bir tahmin performansı sağlayabilmek için birden fazla modeli kullanan bir yaklaşımdır. Birleştirme yöntemlerinin temel prensibi, bir yöntemi tek bir biçimde kullanmak yerine farklı model oluşturma yöntemlerinin doğrusal bir kombinasyonunu kullanmaktır. Bu yöntemlerdeki temel amaç, tahmin performansını artırmaktır.

Başarılı bir birleştirme süreci için öğrenicinin çeşitli olması önerilir. Çünkü modeller arasında önemli farklılıklar olduğu zaman daha iyi performans gözlenmektedir. Ancak yanı sıra eğitim verileri tek bir modele kıyasla daha yüksek aşırı uyum riskine sahip olduğu için model birleştirme yöntemleri dikkatli uygulanmalıdır.

Birleştirme yöntemleri temelde farklı düşünceleri de değerlendirerek karar almayı baz alır. Örnek olarak; bir yatırım fikri için farklı uzmanlara danışılması, bir ürün satın alınırken ürünü kullanan farklı kişilerin yorumlarının alınması ve bir hastalık üzerine farklı doktorlardan görüş alınması verilebilir.

Birleştirme yöntemlerinde bir problemi çözmek için birden fazla öğrenici eğitilmektedir. Eğitim esnasında karar ağacı, yapay sinir ağları gibi öğrenme algoritmaları kullanılır. Aynı tür öğreniciler tek bir öğrenme algoritmasıyla, farklı tür öğreniciler çoklu öğrenme algoritmalarıyla eğitilir. Birleştirme yöntemleri sayesinde tahmin yeteneği zayıf olan öğreniciler, tahmin yeteneği güçlü olan öğrenicilere yükseltilmektedir. Temel işleyiş, zayıf öğrenicileri üretmek ve sonrasında onları birleştirmek üzerinedir.

Bu amaçla en sık kullanılan birleştirme yöntemleri torbalama ve güçlendirme olmaktadır. Torbalama yöntemi varyansı, güçlendirme yöntemi ise yanlılığı azaltmaya yönelik bir yaklaşımdır. Yanlılık ve Varyans ilişkisine 3.4.3 Yetersiz Uyum başlığı altında yer verilecektir. Bu çalışmada güçlendirme yöntemi kullanılarak karar ağacının tahmin performansının artırılması hedeflenmiştir. Bu nedenle bir sonraki başlıkta güçlendirme algoritması ve karar ağaçlarındaki kullanımına yer verilecektir.

2.6.3. Güçlendirilmiş Karar Ağacı Regresyonu

Schapire'in 1989 yılında zayıf öğrenicilerin güçlü öğrenicilere yükseltilebileceğini kanıtladığı bir çalışmada güçlendirme yaklaşımı ilk kez ortaya atılmıştır. Devam eden yıllarda bu alanda yürütülen çalışmalar sonucunda literatüre bir çok güçlendirme algoritması eklenmiştir.

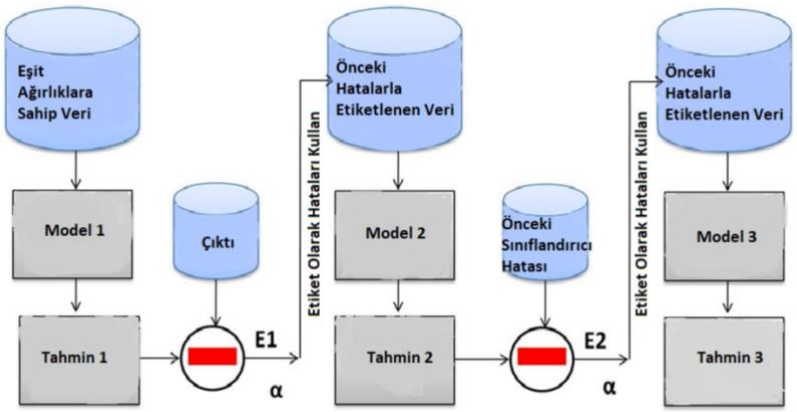
Başlıca güçlendirme algoritmaları; Adaboost, Gradyan Güçlendirme, Light GBM (Gradient Boosting Machine) , CatBoost olmaktadır. Bu çalışmada karar ağacı regresyonu, Microsoft

Azure Machine Learning Studio üzerinde MART(Multiple Additive Regression Trees) Gradyan Güçlendirme Algoritması kullanılarak modellenmiştir.

Güçlendirilmiş karar ağaçları üzerine 2001 yılında ilk kez Friedman, kayıp fonksiyonunun regresyon ağaçlarındaki gradyan inişine (gradient descent) dayanarak tahminde bulunmuştur. Böylece gradyan güçlendirme algoritmasının temeli atılmıştır. Gradyan güçlendirme algoritması, hatayı minimize etmek için gradyan iniş algoritmasını kullanır ve regresyon ağaçlarının bir araya gelmesi ile oluşur.

Güçlendirilmiş karar ağacı regresyonu, danışmanlı bir makine öğrenmesi tekniğidir. Amaç, oluşturulan karar ağacındaki hedef değişkeni, güçlendirmeyi kullanarak tahmin etmektir. Güçlendirme, her ağacın kendinden önce gelen ağacın hatalarını yerleştirerek öğrenmesi ile gerçekleşir.

Gradyan güçlendirme algoritmasında her bir adımdaki hatayı ölçmek ve bir sonraki adımda düzeltmek için bir kayıp işlevi kullanır. Dolayısıyla, tahmin modeli aslında kendinden önceki daha zayıf modellerin bir bütünüdür. Bu şekilde güçlendirme algoritmasıyla adım adım bir dizi regresyon ağacı oluşturulur. Daha sonra türevlenebilir bir kayıp fonksiyonu kullanarak en uygun ağaç seçilir.



Şekil 2-8 Gradyan Güçlendirme Aşamaları

Güçlendirme algoritması, sınıflandırılması zor olan örneklere odaklanır. Şekil 2.8'de (Yangın, 2019) görüldüğü gibi öncelikle ilk veri setinden bir karar ağacı oluşturulur ve tek bir model üretilir. Bu modelde eğitim veri setinden rastgele örnekler oluşturulur ve örneklere eşit

ağırlıklar atanır. Tahmin ile çıktı arasındaki hata bulunur. Bu model üzerinde yanlış sınıflandırılmış örneklerin ağırlıkları artırılırken, doğru sınıflandırılmış örneklerin ağırlıkları azaltılır. Birinci modelin başarısız olduğu örneklerin ağırlıkları daha çok artırılarak ikinci model oluşturulur. İkinci modelin başarısız olduğu örneklerin ağırlıkları daha çok artırılarak da üçüncü model oluşturulur. Böylece her iterasyonda sınıflandırılması zor olan örneklere daha çok ağırlık verilmiş olunur. Modeldeki hata oranı istenen seviyeye inene kadar bu süreç devam eder.

2.7.VERİ MADENCİLİĞİ VE METİN MADENCİLİĞİ

Dijital veri toplama ve depolama teknolojisindeki hızlı ilerlemeler, büyük miktarda veri tabanlarının da toplanabilmesini sağlamıştır. Terabayt veya petabayt büyüklüğünde veriler artık mevcuttur. Ancak bu veri tabanlarından yararlı bilgiler çıkarmak büyük bir zorluktur. Çok sayıda veri içerisinden değerli verileri bulma süreci, madencilik terimi ile tanımlanır.

2.7.1. Veri Madenciliği

Veri madenciliği, büyük veri havuzlarındaki yararlı bilgileri otomatik olarak bulma işlemi şeklinde tanımlanır. Veriler, yapılandırılmış bilgiler veya sayılar olarak ifade edilebilir (Dafni, 2014). Veri madenciliği genellikle üç ana süreçten oluşur. Bu süreçler, verilerin hazırlanması, verilerin incelenmesi ve verilerin modellenmesi olmaktadır.

Verilerin hazırlanması, veri madenciliğinde en çok zaman alan süreçtir. İyi bir model oluşturmak için iyi hazırlanmış verilere sahip olmak gerekir. Amaç hasarlı, aykırı veya gürültülü (tutarsız ve hatalı) verileri ortadan kaldırmaktır. Bu aşamada yapılabilecek çalışmalardan biri olan kayıp veri temizliği, 3.1.3. Kayıp veri temizliği başlığı altında açıklanmıştır.

Bu süreçte veri örüntüleri analiz edilir, temsili veriler bulunur ve trend dışı veriler ortadan kaldırılır. Trend dışı veriler, normal modeli olumsuz etkileyebileceğinden, kaldırılması etkili olabilir. Daha sonra, verilerin bazı hesaplamalar ile belirli bir biçime dönüştürülmesi gerekir. Örnek olarak bir yapay sinir ağı uygulamasında girdi verilerinin normalleştirilmesi verilebilir. Hesaplamalardan sonra da sonuçların ilk veri formuna dönüştürülmesi gerekir. Yeni verilerin modele geldikçe ayarlanması önemlidir, aksi takdirde model çalışmayabilir.

Bir sonraki adım olarak, incelenecek ve sonunda optimize edilecek olan veriler üzerine araştırma yapılır. Bu araştırma, verilerin yapısı ve genel ilişkileri hakkında bilgi verir. Veri madenciliği sürecindeki son adım ise verilerin modellenmesidir.

Yukarıda belirtilen üç ana süreç, analiz ve modellemeyi kolaylaştırmak için daha fazla adıma da ayrılabilir. Veri madenciliği sürecinin iyi yönetilmesi, projenin başarısı için önem taşımaktadır. Veri madenciliğinde kullanılan başlıca proje yönetim sistemleri CRISP-DM (Cross-Industry Standard Process for Data Mining), SEMMA (Sample, Explore, Modify, Model, and Assess) ve KDD (Knowledge Discovery in Databases) olmaktadır.

2.7.2. Veri Madenciliği ile Metin Madenciliği İlişkisi

Sayısal veya mantıksal özelliklere sahip veri tabanları gibi iyi yapılandırılmış verilerden, ilgi çekici kurallar veya kalıplar keşfetmek için çeşitli veri madenciliği araştırmaları vardır. Ancak veri madenciliği tekniklerini doğrudan bir metne uygulamak zordur. Çünkü ham bir metin, heterojen verilerden ve yapılandırılmamış veya yarı yapılandırılmış verilerden oluşabilir.

Metin madenciliği, haber makaleleri, teknik makaleler, kitaplar, dijital kütüphaneler, e-posta mesajları, bloglar, raporlar, web sayfaları, Microsoft Word, Html ve Pdf belgeleri gibi keşfedilecek bilgileri açıkça içermeyen metinlerden yeni bilgiler keşfedilmesini sağlar (Djouadi ve diğ, 2007).

Metin madenciliği, yapılandırılmamış veya zayıf yapılandırılmış metinlerin, öznitelikler tarafından tanımlanan örnek bir tablo halinde yapılandırılmış verilere dönüştürülmesini sağlayan özel prosedürlerle, genel veri madenciliğinden ayırt edilebilir. Modelleme aşamasında, bu tür bir örnek tablo, standart veya uyarlanmış veri madenciliği algoritmaları tarafından, verilerde gizlenen bilgileri ortaya çıkarmak için kullanılabilir. Metin madenciliği, metin sınıflandırma ve kümeleme, özetleme ve sentiment (duygu/duyarlılık) analizi gibi çalışmalarla metin kaynaklarından model ve kalıpların oluşturulmasıyla ilgilenen bir araştırma alanıdır. Metin madenciliği veya metin analitiği olarak da bilinen bu araştırma alanı, genellikle veri madenciliği araştırmalarının bir alt alanı olarak kabul edilir (Perovsek ve diğ, 2016).

2.7.3. Metin Madenciliği

Metin madenciliği, büyük miktarda metin verilerinden örtük, önceden bilinmeyen ve potansiyel olarak yararlı bilgilerin çıkarılması olarak tanımlanır. Bir metinde akıllı metin analizi veya bilgi

keşfi olarak da bilinen metin madenciliği, yapılandırılmamış bir metinden ilgi çekici ve önemsiz olmayan bilgilerin çıkarılması sürecini ifade eder. Yeni bilgileri keşfetmek için metinsel verilerin otomatik ve yarı otomatik yollarla araştırılması ve analiz edilmesiyle gerçekleşir.

Metin madenciliğinde metin, belge ve terim olmak üzere iki temel birimden oluşur. Belge genellikle bölümler, paragraflar ve hatta e-posta mesajları gibi metinlerden oluşur. Terim ise bir belgedeki sözcük, sözcük çifti veya ifade olarak tanımlanır.

Metin madenciliği süreci, metin ön işleme, metin dönüşümü (öznitelik oluşturma), öznitelik seçimi, ve örüntü keşfi ve değerlendirme olmak üzere temelde 4 aşamadan oluşmaktadır.

Metin ön işleme adımı, noktalama işaretlerinin kaldırılması, gereksiz kelimelerin çıkarılması, kelimelerin temel formda gösterilmesi gibi işlemleri içerir. Metin ön işleme tekniklerine ilerleyen bölümlerde detaylı şekilde yer verilecektir.

Bir sonraki süreç, metin dönüşümüdür. Metin dönüşümü, metin belgelerinden faydalı bilgi edinmenin çok önemli bir parçasıdır. Ana görevi, yapılandırılmamış verilerin iyi yapılandırılmış bir veri temsiline dönüştürülmesidir. Metin dönüşümü adımında, metin belgelerinden bir dizi öznitelik oluşturulmaktır. Belirli bir belge grubu için seçilen tüm öznitelikler kümesine temsili model denir. Her belge, seçilen temsili modelin her özniteliği için birer sayısal değer olacak şekilde sayısal bir vektörle temsil edilir. Bu yapı kullanılarak, bir vektörün her sayısal bileşeninin bir öznitelikle ilgili olduğu ve seçilen belgedeki özniteliğin önemiyle ilgili ağırlığı temsil ettiği, öznitelik vektörleri adı verilen en standart metin madenciliği belgesinin temsili elde edilir. Genellikle bu tür metin tabanlı öznitelik vektörlerinde ağırlıklar oldukça aralıklıdır, çünkü ağırlıkların çoğunluğu sıfıra eşittir. Metin dönüşümü sürecinin amacı, belirli bir belge grubundaki her belge için yüksek kaliteli bir öznitelik vektörü çıkarmaktır (Perovsek ve diğ, 2016).

Metin belgeleri çeşitli şekillerde temsil edilir. Temsillerden biri kelime çantasıdır(Bag of words). Bu yaklaşımda kelimeler, bulundukları sıraya bakılmaksızın bağımsız olarak değerlendirilir. Bu teknikte her sözcük, belge alanında bir boyut olarak temsil edilir. Belge de bir vektör olarak temsil edilir. Her kelimenin ağırlığı, bulunuş sıklığına göre hesaplanır. İlgili doküman kümeleme, sınıflandırma, özetleme gibi işlemler için çeşitli matris ayrıştırma

tekniklerinin uygulanmasını sağlayan kelime çantası temsilinde bir matris formatına dönüştürülür. İki belge arasındaki benzerlik derecesi, kosinüs benzerliği, öklid mesafesi gibi mesafe / benzerlik ölçüleri kullanılarak hesaplanır. Kelimeler, ağırlıklarına göre sıralanır ve en yüksek skorlu kelimeler kullanılır.

Belge gösteriminin başka bir yöntemi, vektör uzay modelidir. Bu yöntem, belgelerin anlamsal benzerliğini temsil eder. Belgeler vektörler olarak gösterilir. Belgeler arasındaki benzerlik, vektör aritmetik işlemleri kullanılarak hesaplanır. Terimlerin ağırlıkları Terim Frekansı ve Ters Belge yöntemi kullanılarak hesaplanır. Belgeler arasındaki benzerlik değerleri 0 ile 1 arasında sınırlandırılmıştır. En benzer belgelerin kosinüs değeri 1'e yakındır ve benzer olmayan belgeler 0'a yakın değerlere sahiptir.

Metin madenciliği sürecinin sonraki aşaması öznitelik seçimidir. Öznitelik seçimi, bazı ölçütlere göre orijinal kümeden en faydalı ve en önemli öznitelikleri seçerek veri kümesindeki öznitelik sayısının azaltılmasını amaçlamaktadır. Öznitelik seçim yöntemleri, sınıf etiketi bilgilerinin gerekip gerekmediğine bağlı olarak denetlenen ve denetlenmeyen olmak üzere ayrılır. Bilgi Kazancı ve Ki-Kare istatistiği, denetlenen iki yöntemdir. Belge sıklığı, Terim Gücü, Entropi tabanlı sıralama ve Terim Katkısı, öznitelik seçimi için kullanılan dört denetimsiz yöntemdir.

Öznitelik seçiminden sonra, yeni örüntüleri keşfetmek için çeşitli metin madenciliği yöntemleri uygulanır. Bu yöntemlere örnek olarak metin kategorizasyonu, metin kümelemesi, kavram-varlık çıkarımı, duyarlılık analizi, belge özetleme ve varlık-ilişki modellemesi verilebilir. Bulunan örüntüler son kullanıcıya sunulur. Daha sonra hassasiyet, hatırlama, F-skoru gibi çeşitli metrikler kullanılarak değerlendirilir (Dafni, 2014).

3. MALZEME VE YÖNTEM

Bu çalışmada yazılım proje yönetiminde efor tahmininin otomatik bir sistem kullanılarak derin öğrenme tabanlı bir yöntemle gerçekleştirilebileceği bir yöntem sunulmuştur. Bu sayede tahminlerin insan yargısına bağlı olası tutarsızlıklarının, tahmin sürecinde sarfedilen eforun ve tahmin süresinin azaltılması amaçlanmıştır. Uygulama aşamasında derin öğrenme ve karar ağacı tabanlı makine öğrenmesi teknikleri karşılaştırılmıştır. Modellerin performansını artırmak amacıyla öğrenme süreci öncesinde veri seti bir dizi hazırlık sürecinden geçirilmiştir.

Bu bölümde veri hazırlığı aşamasında uygulanabilecek işlemler, hiperparametre ayarlama yöntemleri ve çalışmada uygulanan modellerde kullanılabilecek temel hiperparametreler detaylı şekilde anlatılmıştır. Son olarak model performanslarının değerlendirilebileceği temel metrikler açıklanmıştır.

3.1.VERİ HAZIRLIĞI

Çalışmada kullanılacak veri setinin öğrenme süreci öncesinde bir hazırlık sürecinden geçmesi, hesaplama süresinde ve maliyetinde tasarruf, uygulama performansında artış gibi önemli faydalar sağlayabilmektedir.

Bu amaçla uygulamada kullanılacak veri setleri, yapılarına ve uygulamanın amacına bağlı olarak çeşitli veri ve metin madenciliği yöntemleri yardımıyla öğrenme sürecine hazırlanabilir.

Bu yöntemler, projeye katkısı olmayacak verilerin çıkarılması ya da farklı formlarda daha verimli kullanılabilecek olan verilerin ilgili forma dönüştürülmesi şeklinde olabilir. Veri hazırlığı kapsamında yapılabilecek etkili bazı çalışmalar bu bölümde anlatılacaktır.

3.1.1. Metin ön işleme

Temel anlamda metin ön işleme, n-gramda kullanılacak kelime dağarcığını en aza indirerek modelin doğruluğunu artırmayı hedefleyen işlemler dizisidir. N-gram, metinde bulunan bitişik n kelimenin tüm kombinasyonlarından oluşan bir dil modelidir. N-gram sayısını sadeleştirmek uygulamanın hızını ve verimini artırmaktadır. Aşağıda metin ön işleme örneklerine detaylı şekilde yer verilmiştir.

- Dil seçiminin yapılması:

  Metin ön işleme sürecinde yapılan bazı işlemler dil bağımlı olmaktadır. Bu nedenle bu işlem bir yazılım aracı ile yapılıyorsa en başta dil seçiminin yapılması önem taşımaktadır.
- Gereksiz kelimelerin çıkarılması:

  Dilbilgisi olarak anlamlı olsa da edat, bağlaç gibi makine öğrenmesi için bilgi içermeyen kelimelerin çıkarılması etkili bir metin ön işleme yöntemidir. Dil seçiminin İngilizce olduğu bir duruma örnek olarak a, about, above, the kelimeleri verilebilir.
- Kelime köklerinin ayrılması:

  Birden fazla kelime formunu temsil etmek için tek bir standart form tanımlama işlemidir. Benzer kelimelerin oluşum sayısını azaltmak için kullanışlıdır. İngilizce cümle yapısından örnek olarak "Swimming is good for building muscle" cümlesindeki kelimelerin "swim be good for build muscle" olarak dönüştürülmesi verilebilir. Böylece n-gram kelime dağarcığı azaltılarak daha kullanışlı hale getirilir.
- Fiil daraltmalarının genişletilmesi:

  Metnin diline bağlı olmak üzere bazı dillerde fiil daraltmaları kullanılmaktadır. İngilizce için örnek olarak would not yerine wouldn't daraltması verilebilir. Bu gibi daraltmalar içeren kelimeler, n-gram modülünde daha sağlıklı işlenebilmesi için genişletilebilir.
- \\ işaretlerinin / işaretine normalleştirilmesi:

  Özellikle kodlama içeren metinlerde sıklıkla görülen \\ işareti, / işaretine dönüştürülebilir (Microsoft Azure Documentation, 2020).
- Özel karakterlerle birleştirilmiş metinlerin ayrılması:

  & - gibi özel karakterlerle birleştirilmiş metinler uygulayıcının tercihine bağlı olarak ayrılabilir. Örnek olarak Macbook-Air kelimesinin 2 ayrı kelime olarak ayrılması verilebilir. Ancak çok kelimeli ifadelerin kendine özgü doğası olabilir. Bu tür çok kelimeli ifadelerin anlamı, tek tek kelimelerden pek çıkarılamaz (Collados ve diğ, 2017).
- Birleşik Kelimeleri ayırmak:

  Bazı diller sıklıkla iki farklı kavramı tanımlayan iki kelimeyi birleştirir. Bu gibi durumlar, metin çözümleme tekniklerinin etkinliğini etkiler. Çünkü bir kavram birçok unik sözcükte gizlenebilir veya iki kelimeye bölünebilir. Bu nedenle uygulayıcı bu kelimeleri ayırmak isteyebilir. Almanca, Fince, Danca, Felemenkçe, Norveççe, İsveççe ve Yunanca gibi dillerde bir işaretle ayrılmayan kapalı birleşik kelimeler yer almaktadır.

- Segmentasyon:

  Çince, Japonca ve Laoca gibi bazı dillerde kelimeler arasında boşluk yoktur. Bu nedenle kelimeye dayanan metin çözümleme teknikleri, kelimeleri doğal olarak ayrı birimlere ayıramaz. Belgelerin istatistiksel bir program tarafından işlenebilmesi için otomatik segmentasyon kullanılmaktadır (Lucas ve diğ, 2015).

3.1.2. Öznitelik Karması:

Metin veri madenciliğindeki önemli aşamalardan biri öznitelik(özellik) dönüşümüdür. Öznitelik karması, metin verilerini tamsayı olarak temsil edilen bir dizi özniteliğe dönüştürür. Öznitelik karması, yüksek boyutlu girdi alanını düşük boyutlu alana eşleyerek sınıflandırma algoritmalarını ölçeklendirmeyi amaçlayan etkili bir boyutsal azaltma yöntemidir.

Öznitelik karması, öznitelik sayısını azaltmak ve işlem süresini hızlandırmak için etkili bir yaklaşımdır. Bu yöntemde modeli eğitmek için n-gramlık kelimeleri veya harfleri, öznitelik vektörleriyle eşleştirmek için karma (hash) fonksiyonları kullanılmaktır. Yüksek boyutlu alan daha düşük boyutlu bir alana, karma fonksiyonunu kullanmak suretiyle azaltılır. Diğer bir deyişle, birden fazla özelliğin aynı karma anahtarına adreslenebildiği durumlarda, öznitelikleri birleştirerek bu adreslemeyi yapar ve sayılarını toplar.

Öznitelik karması, ünik kelimeleri tamsayılara dönüştürerek çalışır. Bir n-gram sözlüğü oluşturulur ve sözlük terimleri karma değerlerine dönüştürülür. Metin verileri satırlar, karma öznitelikleri sütunlar olacak şekilde bir tablo oluşturulur. Bu tabloda sütun değeri 0 olan satırlar karma özelliğini içermiyordur; 1 olan satırlar ise içeriyordur.

Her bir öznitelik karması, bit sayısına (k) ve n-gram sayısına bağlı olarak bir veya daha fazla N-gram metin özelliği temsil eder. Öznitelik karması konusunun daha iyi anlaşılabilmesi için bu kapsamda yer alan temel kavramlardan Karma boyutları, N-gram ve Vowpal Wabbit bu bölümde alt başlıklar halinde anlatılacaktır.

Bu yöntem, öznitelik sayısını çok yüksek boyutlardan (ör. 2^{26}) orta boyutlara kadar (ör. 2^{16}) azaltmada çok etkili olmasına rağmen, önemli miktarda bilgi kaybına da neden olabilir. Bu bilgi kaybı durumları özellikle çok farklı sınıf dağılımlarına sahip yüksek sıklıkta görülen öznitelikler arasında karma çatışmaları olduğu zaman görülür (Caragea ve diğ, 2012).

Karma çatışmaları çoklu n-gramların aynı karma anahtarlara eşlenmesi esnasında olmaktadır. Bu sorun, az veri olması durumunda daha büyük karma tablolar kullanılarak ve bit boyutu artırılarak önlenebilir. Bit boyutu, karma tablosu oluşturulurken bitlerin sayısını belirler. En uygun bit boyutunu seçme kararı, n-gram kelimelerin boyutuna bağlıdır (Mehdiyev ve diğ, 2017).

Öznitelik karması, haber kategorizasyonu, spam filtreleme, sosyal ağlarda duygu analizi gibi farklı alanlarda başarılı uygulamalar bulmuştur. Bu yöntemin başlıca avantajları, değişken uzunluktaki metin verilerini eşit uzunluktaki sayısal öznitelik vektörlerine dönüştürmesi, boyutsal küçültmeyi sağlaması ve öznitelik ağırlıklarının daha hızlı elde edilmesine olanak tanımasıdır. Çıktıların sayısal olması, sınıflandırma, kümeleme, regresyon gibi birçok farklı makine öğrenme yönteminin kullanılmasını mümkün kılar. Buna karşılık metin sütunu, eğitim için eğer olduğu gibi kullanılırsa, çok farklı değerler içeren kategorik bir öznitelik sütunu olarak işlem görür.

- **Karma Boyutları:**

Veriler için bir boyutun yeterli olup olmadığı, eğitim metnindeki n gramlık kelime dağarcığının boyutuna bağlıdır. Geniş kelime dağarcığında daha fazla alana ihtiyaç duyulabilir. Boyut yeterli olmadığında karma eşlenmesi esnasında oluşabilecek karma çatışması nedeniyle bilgi kaybı riski söz konusudur.

Kullanılan bit sayısı da öznitelik boyutunu doğrudan etkiler. Örneğin bit sayısı 14 ise 2^k formülü üzerinden hesaplandığında karma tablosu 2^{14} = 16.384 giriş alabilir.

Girdi olarak birden fazla metin sütunu seçilmesi, öznitelik boyutu üzerinde büyük etki oluşturabilir. Örneğin, tek bir metin sütunu için 10 bitlik karma kullanılırsa, çıktı 2^{10} =1024 sütun içerir. İki metin sütunu için 10 bitlik karma kullanılıyorsa, çıktı iki katı yani 2048 sütun içerir.

Birçok problem için 10 bitlik karma tablosu fazlasıyla yeterli olmakla birlikte eğitim metnindeki n-gram kelime dağarcığının boyutuna bağlı olarak karma çatışmalarını önleyebilmek için daha fazla alan gerekebilir.

- **N-gram:**

N-gram, kaynak metinde bulunan bitişik n kelimenin tüm kombinasyonlarından oluşan bir dil modelidir. Kelime yerine fonem, hece veya harf gibi versiyonları da olabilir. N-gram kavramı, harf üzerinden aşağıdaki şekilde örneklendirilebilir.

Örneğin E = {A, F, G, C, L, B, A, D, A, M} olaylar dizisinde, bigram (2 gram) özniteliklerinin tümü {AF, FG, GC ..., AM}; trigram (3 gram) öznitelikleri de {AFG, FGC, CLB ..., DAM} olmaktadır. Bu şekilde önceden tanımlanmış boyutların n-gram kombinasyonunun ele alındığını varsayalım. 5 gram (unigramlar (1 gram), bigramlar (2 gram), trigramlar (3 gram), quadgramlar (4 gram) içerecek şekilde) girdi öznitelik alanı boyutu ve 15 benzersiz olay büyüklüğünde bir alfabe olması durumunda tüm olası n-gramların sayısı 15 tekil harfin 1'li, 2'li 3'lü 4'lü ve 5'li kombinasyonlarının toplamı şeklinde olacaktır.

$N_{toplam_öznitelik} = 15 + 15^2 + 15^3 + 15^4 + 15^5 = 813.630$

N-gram yöntemi, sadece harfleri kodlamakla kalmayıp aynı zamanda otomatik olarak sıralayabilmesi nedeniyle de çok etkilidir. Bununla birlikte, yukarıdaki örnekte görülebileceği gibi, n-gram temsillerinin en büyük dezavantajı, sınıflandırma problemi için üretilen girdi özniteliklerinin büyük olma eğilimidir. Bu değişkenlerin sayısı, alfabe boyutu ve n-gram uzunluğu ile katlanarak artar. Üretilen tüm özniteliklerin kullanılması, yüksek hesaplama maliyetleri ve düşük doğruluk sağlayacak şekilde tahmin sistemine aşırı yüklenilmesine neden olacağından, dikkatle kullanılması önem taşımaktadır (Mehdiyev ve diğ, 2017).

- **Vowpal Wabbit:**

Ölçeklenebilir doğrusal çözücülere olan ticari ilgi nedeniyle, bir dizi şirket yüksek derecede optimize edilmiş kütüphaneler geliştirmiştir ve bazıları halka açık hale getirilmiştir. Öznitelik karması tekniğinde Vowpal Wabbit, Tensorflow gibi kütüphaneler kullanılmaktadır. Bu bölümde öznitelik karması modülü tarafından en sık kullanılan Vowpal Wabbit kütüphanesi anlatılacaktır.

Vowpal Wabbit, açık kaynaklı hızlı bir makine öğrenmesi kütüphanesidir. Stokastik gradyan iniş öğrenme yöntemini kullanan bir çevrimiçi öğrenme yazılımıdır. N-gram gibi doğrusal olmayan öznitelikler üzerinde son derece hızlı öğrenme sağlar. Tüm eğitim verilerini belleğe okutmak zorunda kalmaz. Bu sayede ham veriler çok hızlı bir şekilde yinelenir ve diğer

modellerden daha hızlı bir tahminleyici geliştirebilir. Vowpal wabbit CPU hızı yerine disk giriş / çıkış hızına bağlıdır. Düşük bellek talebi, hesaplama verimliliği ve yakınsama hızı gibi avantajlara sahiptir.

Vowpal Wabbit yazılımında stokastik gradyan iniş uygulaması, projelerdeki modeli ve öznitelikleri değerlendirmek için kullanılabilmektedir. Stokastik gradyan kökenli vowpal wabbit uygulaması için hiperparametreler, öğrenme oranı, eğitim veri setinden geçiş sayısı, eğitim örneklerinin uygulayıcıya verilme sırası ve L1 ve L2 regülasyon sabitleri bulunur (William, 2014).

Vowpal Wabbit, murmurhash3 adı verilen bir karma fonksiyonunu kullanarak kelimeleri bellek içi dizinlere ekler. MurmurHash, kripto-grafik olmayan bir karma fonksiyonudur. Murmurhash3, Murmurhash algoritmasına dayanan karma fonksiyonları arasında en popüler olanlardan biridir.

3.1.3. Kayıp Veri Temizliği

Tahmin projelerinde veri kalitesi oldukça önemli bir unsurdur. Veri kümelerinin kalitesi, sadece tutarlılıkla değil, aynı zamanda eksiksizlikle de ilgilidir. Geliştiricilerin proje yönetimi kapsamına uygun plan veya faaliyetler gerçekleştirebilmesi için yazılım efor tahmininin sağlıklı yapılması oldukça önemlidir. Bunun için de tahmin sürecinde tüm projenin geçmiş efor verisine sahip olunması önem taşımaktadır. Eksik verilerin olması durumunda bu verilerin doğru bir ön işleme sürecinden geçmesi, gelecek efor tahminleri üzerinde önemli bir etki yaratacaktır.

Yazılım efor tahmin uygulamalarında düzenli olarak sorulan birkaç soru vardır. Hangi veriler kullanılmalıdır? Hangi veri kümelerinin kullanılması daha güvenilirdir? Gelecekteki bir projeyi tahmin etmek için belirli bir proje nasıl seçilir? Bu sorular geçmiş verileri analiz etmeden önce araştırmacıların karar vermeleri için önemlidir. Çoğu araştırmacının belirli çalışmaların arasından nasıl seçim yapılacağı konusunda açık yollar göstermediği gözlenmiştir. Yazılım efor tahmini sürecinde, veri eksikliği ve veri türleri konularındaki bilgi eksiklikleri nedeniyle zorluklarla karşı karşıya kalınabilmektedir (Arbain ve diğ, 2019).

Bu tür durumlara çözüm olması için veri temizleme süreci, veri madenciliğinin en önemli aşamalarından biri olarak karşımıza çıkmaktadır. Temel olarak 3 tür eksik(kayıp) veri vardır:

Rastgele Eksik Veri(MAR), Rastgele Olmayan Eksik Veri(MNAR) ve Rastgele Tamamen Eksik Veri(MCAR). Bir değişkende eksik veri olma olasılığı, değişkenin kendisine bağlı olmayıp diğer değişkenlere bağlı ise bu tür veriler Rastgele Eksik Veridir. Örneğin, bir anket çalışmasında anketörün soru sorması ancak yanıtlayıcının bilerek cevap vermemesi. Bir değişkende eksik veri olma olasılığı, değişkenin kendisine bağlı ise bu tür veriler Rastgele Olmayan Eksik Veridir. Örneğin, anketörün bir soruyu yanlış sorması nedeniyle doğru cevaba ulaşılamaması. Rastgele Tamamen Eksik Veri, eksik verilerin diğer değişkenlere, hatta kendisine de bağımlı olmadığı zamandır. Veriler basitçe eksiktir ve eksik verilerin nedenini bulmak için değişkenler arasında hiçbir ilişki türetilemez. Örneğin, anketörün soruları unutması.

En düşük hata değerine ulaşmak için eksik verilere yönelik temel olarak iki yaklaşım vardır. Eksik verileri tahmin etmek ya da silmek (Gharipour ve diğ, 2010). İlk yaklaşımda eksik veriler tahmin edilebilir ya da bu veriler yerine ortalama, mod, medyan gibi değerler yazılabilir. Diğer yaklaşımda ise bu tezde olduğu gibi çok az sayıda eksik veri varsa silmek daha makul bir yaklaşım olmaktadır.

3.2.HİPERPARAMETRE AYARLAMA

Derin öğrenme algoritmaları, algoritmanın davranışını kontrol eden birçok hiperparametreye sahiptir. Hiperparametrelerin, bir modelin eğitim sürecine, kalitesine, doğru sonuçlar üretme kabiliyetine ve performansına etkileri oldukça büyüktür. Bunun sonucu olarak bir algoritmayı çalıştırmanın zaman ve bellek maliyeti ile hiperparametrelerin ilişkisi göz ardı edilemez.

Hiperparametre ayarlama(Hyperparameter Tuning), veri setine göre hangi ayarların daha iyi performans sağladığını belirlemek anlamına gelmektedir. Derin öğrenmede hiperparametreler sezgisel olarak manuel veya grid aramayla ayarlanır.

Bir modelin hiperparametrelerini seçmek için temelde manuel veya otomatik olmak üzere iki farklı yaklaşım kullanılabilir. Hiperparametreleri manuel olarak seçebilmek için hiperparametrelerin ne işe yaradığı ve derin öğrenme modellerinin iyi bir genellemeyi nasıl elde ettiği hakkında bazı bilgiler gereklidir. Hiperparametreleri otomatik olarak seçmek için daha az bilgiye ihtiyaç duyulur ve hesaplama maliyeti manuel ayarlamadan daha düşüktür.

3.2.1. Manuel Hiperparametre Ayarlama

Hiperparametreleri manuel olarak ayarlamak, hiperparametreler arasındaki ilişkiyi, genelleştirme hatasını ve eğitim sürecini iyi anlamayı gerektirir. Manuel hiperparametre ayarlamada temel amaç, modelin etkin kapasitesini görevin karmaşıklığına uyacak şekilde ayarlamaktır.

Etkili kapasitenin 3 kısıtlaması vardır. Bunlar, öğrenme algoritmasının kayıp fonksiyonunu en aza indirme yeteneği, modelin temsil kapasitesi ve kayıp fonksiyonu ve eğitim sürecinin modeli ne derece genelleştirdiğidir.

Modelin etkin kapasitesini kontrol eden en önemli hiperparametre, öğrenme oranıdır. Bir optimizasyon problemi için doğru öğrenme oranı kullanılırsa, modelin etkin kapasitesi en yüksek olur. Öğrenme oranı, eğitim hatası için U şeklinde bir eğriye sahiptir. Çok yüksek bir öğrenme oranı kullanmak, eğitim hatasının azalmak yerine artmasına neden olabilir. Çok düşük bir öğrenme oranı ise eğitimin yavaşlamasına ve yine yüksek bir eğitim hatasına neden olabilir.

Öğrenme oranı ve diğer hiperparametreleri ayarlarken, eğitim ve test sürecinde yetersiz uyum ya da aşırı uyum olup olmadığının izlenmesi ve kapasitenin buna göre ayarlanması önemlidir. Hiperparametreleri manuel olarak ayarlarken düşük bir eğitim hatasına ulaşılmaya çalışılmalıdır. Bu süreçte hatırlanması gereken bir nokta, genelleştirme hatasının her zaman daha fazla eğitim verisi kullanılarak azaltılabilmesidir.

3.2.2. Otomatik Hiperparametre Ayarlama:

Derin sinir ağları, genellikle fazla sayıda hiperparametrenin ayarlanmasından önemli ölçüde yararlanır. Ancak az sayıda hiperparametrenin ayarlanmasıyla da iyi bir performans gösterebilir. İyi bir başlangıç noktası varsa, manuel hiperparametre ayarlama iyi bir yaklaşımdır. Ancak öyle olmadığı durumlarda, hiperparametreler için faydalı değerler bulmak adına otomatik algoritmanın kullanılması etkili olabilir.

Özellikle üç veya daha az hiperparametre için Grid arama yöntemini kullanmak oldukça yaygındır. Grid aramada kullanıcı, her bir hiperparametre için üzerinde çalışılacak küçük bir sonlu değer kümesi seçer. Değerler listesindeki en küçük ve en büyük değer, benzer denemelerle gerçekleşen önceki deneyimlere dayanarak seçilmelidir. Bu işlem, optimal değerin seçilen

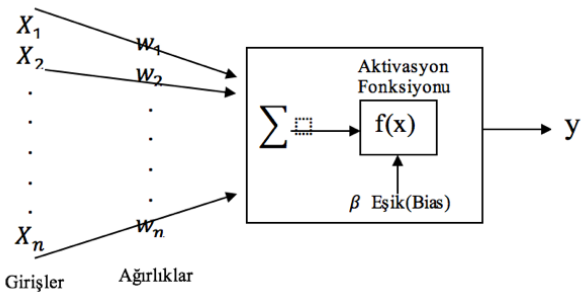
aralıkta olmasını sağlamak içindir. Daha sonra model, her bir ayrı hiperparametre için Grid arama algoritması ile eğitilir. En iyi hiperparametre kombinasyonu, en iyi doğrulama hatasına yol açan denemeden bulunabilir (Persson, 2018).

3.3.DERİN SİNİR AĞI HİPERPARAMETRELERİ

Derin yapay sinir ağlarında hiperparametre seçimi, ağın performansını belirleyen en önemli aşamalardan biridir. Derin sinir ağlarında kullanılan temel hiperparametrelerin kapsamlı bir listesi, bu hiperparametrelerin ağa etkileri ve başarılı bir snir ağı eğitimi gerçekleştirmek için faydalı teknikler bu bölümde detaylı bir şekilde anlatılacaktır.

3.3.1. Aktivasyon Fonksiyonu:

Bir katmanın nöronlarının çıktısını bir sonraki katmana iletmek için aktivasyon fonksiyonları kullanılmaktadır. Her bir nöron, bir diğer nöronun çıktısını veya dışarıdan gelen bir uyarıyı girdi sinyali olarak alır, aktivasyon fonksiyonu ile işler ve bir diğer nörona veya dışarıya verilmek üzere çıktı sinyali üretir. Şekil 3.1'de(Taylor, 2005) genel akış gösterilmektedir. Bu fonksiyonun aktivasyon fonksiyonu adıyla anılmasının sebebi, biyolojik sinir hücresinin aktif olup iletim yapabilmesi için gerekli olan eşik değerini aşabilecek değeri üreten bir fonksiyon olarak düşünülmesidir. Bu anlamda aktivasyon fonksiyonları, nöronun aktivasyonunu sağlayan bir fonksiyondur. Doğrusalsızlığı (nonlinearity) ağın modelleme yeteneklerine dahil etmek amacıyla yapay sinir ağları(YSA) yapısındaki gizli nöronlar için aktivasyon fonksiyonları kullanılmaktadır.



Şekil 3-1 Matematiksel modelleme ile elde edilen bir Yapay Sinir Hücresi

Bir hücreye gelen net girdi, ağırlık değerleri ve girdilerin çarpımlarının toplanması ile hesaplanır. Bu fonksiyona toplama fonksiyonu denmektedir.

$$Net_i = \sum_{i=1}^{n}(w_{ij}x_{i+}\beta) \quad (3.1)$$

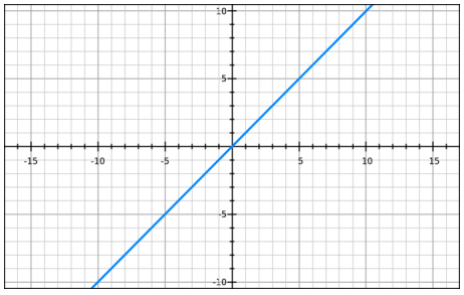
Toplama fonksiyonu sonucunda elde edilen net girdi değeri, doğrusal ya da doğrusal olmayan bir aktivasyon fonksiyonundan geçirilerek nöronun net çıktısı hesaplanır. Örneğin sigmoid bir aktivasyon formülünde toplama fonksiyonu aşağıdaki şekilde yer almaktadır.

$$y = f(Net) = \frac{1}{1+e^{-Net}} \quad (3.2)$$

Pek çok aktivasyon fonksiyonu, grafik halinde gösterildiğinde S'ye benzeyen sigmoidal adında bir lojistik sınıfına aittir (Patterson ve Gibson, 2017). Yapay sinir ağlarında uygulanan temel aktivasyon fonksiyonları aşağıda listelenmiştir.

3.3.1.1. Doğrusal

Doğrusal fonksiyon (bkz. Şekil 3.2) bağımlı değişkenin bağımsız değişkenle doğrudan orantılı bir ilişkiye sahip olduğu f (x) = x fonksiyonudur. Uygulamada fonksiyonun değişmeden sinyali geçtiği anlamına gelir. Yapay sinir ağlarının girdi katmanında bu aktivasyon fonksiyonunun kullanıldığı görülmektedir.



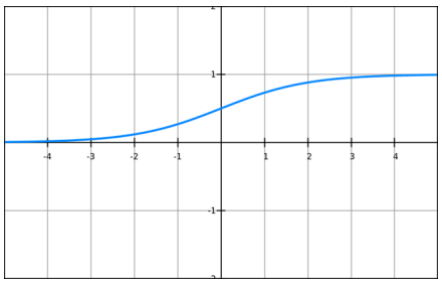
Şekil 3-2 Doğrusal Aktivasyon Fonksiyonu

3.3.1.2. Sigmoid

Sigmoid fonksiyonu, (bkz Şekil 3.3) yakın sonsuz aralığın bağımsız değişkenlerini, 0 ile 1 arasında basit olasılıklara dönüştürmektedir. Bir sigmoid aktivasyon fonksiyonu, her bir sınıf için bağımsız bir olasılık çıkarmaktadır.

$$f(x) = \frac{1}{1+e^{-x}} \quad (3.3)$$

Tüm lojistik dönüşümler gibi, sigmoid de veriyi çıkarmadan aşırı veya aykırı değerleri azaltabilmektedir.

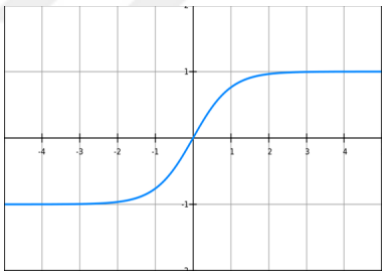


Şekil 3-3 Sigmoid aktivasyon fonksiyonu

3.3.1.3. Tanh

Tanh hiperbolik trigonometrik bir fonksiyondur (bkz Şekil 3.4) Tanjantın sinüsün kosinüse oranını temsil ettiği gibi Tanh, hiperbolik sinüsün hiperbolik kosinüse oranını temsil eder (Tanh (x) = Sinh (x) / Cosh (x))

Sigmoid fonksiyonundan farklı olarak, normalize edilmiş aralık–1 ila 1'dir. Tanh'ın avantajı, negatif sayılar ile daha kolay başa çıkabilmesidir.



Şekil 3-4 Tanh aktivasyon fonksiyonu

3.3.1.4. Hard Tanh:

Tanh'a benzemektedir ancak normalleştirilmiş aralığa uygulanır. 1'den yüksek değerler 1'e, -1'den düşük değerler -1'e dönüştürülür. Bu durum, sınırlı bir karar aralığına izin veren daha sağlam bir aktivasyon fonksiyonuna imkan tanır.

3.3.1.5. Softmax :

Softmax, bir sigmoid fonksiyonu olmakla beraber, verileri sigmoid gibi ikili sınıfa ayırmaz, üçten fazla sınıfa ayırır. Olasılıksal olarak sınıflandırma yaptığı için 0 ile 1 arası bir çıktı olasılığı üretir. Softmax, bir sınıflandırıcının genellikle çıktı katmanında bulunan bir fonksiyondur.

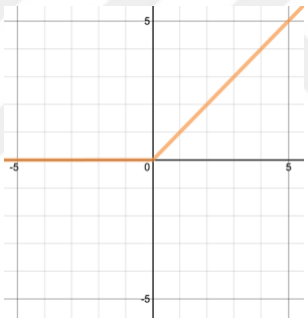
Softmax çıktı katmanının nasıl kullanılacağını daha iyi açıklamak için bir örnek üzerinden ilerleyelim. Çok sınıflı bir modelleme problemi varsa ve bu sınıflar boyunca sadece en iyi skor önemseniyorsa, tüm sınıfların en yüksek puanını almak için argmax fonksiyonuna sahip bir softmax çıktı katmanı kullanılabilir. Softmax fonksiyonu aşağıdaki şekilde hesaplanmaktadır.

$$f(x_i) = \frac{e^{x_i}}{\sum_{i=0}^{k} e^{x_i}} \quad \text{i=0,1,2,..,k} \tag{3.4}$$

Çıktı başına birden fazla sınıflandırma olduğu durumlarda, çıktı katmanı olarak softmax yerine her sınıf için bağımsız bir olasılık sağlayacak şekilde sigmoid fonksiyonu kullanılır.

3.3.1.6. Düzeltilmiş Doğrusal(ReLU)

Düzeltilmiş doğrusal fonksiyon, girdinin yalnızca belirli bir miktarın üzerinde olması durumunda nöronu etkinleştirdiği (f (x) = max (0, x)) bir dönüşümdür. Girdi sıfırın altındayken çıktı sıfırdır. Girdi ancak belirli bir eşiğin üzerine çıktığında çıktı, bağımlı değişkenle doğrusal bir ilişkiye sahiptir (bkz Şekil 3.5).



Şekil 3-5 Düzeltilmiş Doğrusal Aktivasyon Fonksiyonu

Düzeltilmiş doğrusal birimlerin, birçok farklı durumda çalıştıkları kanıtlanmıştır. Sigmoid ve tanh aktivasyon fonksiyonlarına kıyasla ReLU aktivasyon fonksiyonu, kaybolan gradyan(vanishing gradient) probleminden etkilenmez. Araştırmalar, ReLU aktivasyon fonksiyonları kullanılarak derin ağların ön eğitim teknikleri kullanmadan da iyi eğitildiğini göstermiştir.

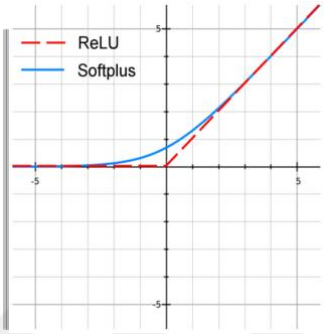
3.3.1.7. Sızıntılı ReLU

Sızıntılı ReLU kullanılarak, nöron ölümü sorunu hafifletilebilir. Aşağıdaki denklemde de görüleceği gibi sızıntılı ReLU ile x <0 olduğunda fonksiyon sıfır olmak yerine küçük bir eğime (örneğin, "yaklaşık 0.01") sahip olacaktır. Sızıntılı ReLU ile pratikte bazı başarılar görülmüştür. Ancak sonuçlar her zaman tutarlı değildir.

$$f(x) = \begin{cases} x & if\ x > 0 \\ 0.01x & otherwise \end{cases} \quad (3.5)$$

3.3.1.8. Softplus

Bu aktivasyon fonksiyonu Şekil 3.6'da gösterildiği gibi ReLU'nun yumuşak bir versiyonu olarak kabul edilir.



Şekil 3-6 ReLU ve softplus aktivasyon fonksiyonlarının görselleştirilmesi

Şekil 3.6, softplus aktivasyon fonksiyonunun (f (x) = ln [1 + exp (x)]) ReLU'ye benzer bir şekile sahip olduğunu göstermektedir. Ayrıca bu grafiğe bakıldığında, ReLU'nun aksine softplus'un sıfır olmayan türevi de görülmektedir (Patterson and Gibson, 2017).

3.3.2. Katman sayıları:

Yapay Sinir Ağı mimarisi Girdi katmanı, Gizli katman ve Çıktı katmanından oluşur.

Girdi Katmanı, dış ortamla iletişim kuran bir katmandır ve gelen verileri değiştirmeden gizli katmanlara yönlendirir. Girdi katmanı, yapay sinir ağına bir model sunar ve çıktı katmanı ona göre model üretir. Girdi katmanı başka bir model sunulduğunda, çıktı katmanı başka bir model üretecektir.

Çıktı katmanı, dış ortama bir model sunar. Çıktı nöronlarının sayısı, yapay sinir ağının çözeceği problemin türü ile doğrudan ilişkili olmalıdır.

Gizli katman, Girdi ve Çıktı katmanı arasındaki ara katmandır. Girdi ile çıktı nöronları arasındaki ilişkinin kurulmasını ve ağ ağırlıklarının belirlenmesini sağlar. Aktivasyon fonksiyonları, mevcut ise gizli katmanda uygulanır. Gizli katmanlar doğrudan dış ortamla

etkileşime girmez, ancak nihai çıktı üzerinde büyük bir etkiye sahiptir. Gizli katman gizli nöronlardan oluşur (Karsoliya, 2012).

Gizli katmandaki nöronların sayısının belirlenmesi, tüm yapay sinir ağı mimarisine karar vermenin çok önemli bir parçasıdır. Gizli katmandaki nöronların sayısı, problem verilerinin karmaşıklığına göre gereğinden fazlaysa aşırı uyum, gereğinden azsa yetersiz uyum meydana gelebilir (Panchal ve diğ, 2011). Gizli katmandaki nöronların sayısını belirlemek için kullanılan başlıca yaklaşımlar aşağıda anlatılacaktır.

- **Deneme ve Hata Yöntemi**

Belirlenen doğruluk seviyesine ulaşıncaya kadar devam eden çeşitli denemelerden oluşur. İki ayrı yaklaşımla uygulanabilir.

İlk yaklaşımda, az sayıda gizli nöron seçilerek başlanır. Genellikle iki gizli nöron seçilir. Sonrasında yapay sinir ağı eğitilir ve eğitimden sonra test edilir. Eğitim ve testler düzelene kadar gizli nöronların sayısı arttırılarak süreç tekrarlanır.

İkinci yaklaşım, ilk yaklaşımın tersidir. Çok sayıda gizli nöronla başlanır. Daha sonra yapay sinir ağı eğitilir ve test edilir. Devamında yavaş yavaş gizli nöronların sayısı azaltılır, tekrar yapay sinir ağı eğitilir ve test edilir. Eğitim ve testler geliştirilinceye kadar bu işlemler tekrarlanır (Sheila ve Deepak, 2013).

- **Başparmak Yöntemi**(Karsoliya,2012)

Başparmak yöntemi, gizli katmanlarda kullanılacak nöron sayısının aşağıdaki koşullarla belirlenmesidir:

- Gizli nöronların sayısı, girdi katmanının nöron boyutu ve çıktı katmanının nöron boyutu arasındaki aralıkta olmalıdır.
- Gizli nöronların sayısı, girdi katmanının nöron boyutunun 2/3'ü ile çıktı katmanının nöron boyutunun toplamı olmalıdır.
- Gizli nöronların sayısı girdi katmanı nöron boyutunun iki katından az olmalıdır.

- **Basit Yöntem**

Girdi çıktı ve gizli nöronların sayısının aynı olma durumudur. Örneğin iki girdi nöronu ve iki çıktı nöronu varsa, konfigürasyon 2-2-2 olmaktadır. 2 girdi nöronu, 2 gizli nöron ve 2 çıktı nöronudur.

- **İki Fazlı Yöntem**

İki fazlı yöntemde sonlandırma koşulu, deneme ve hata metodu ile aynıdır ancak eğitim sürecinde yeni bir yaklaşım söz konusudur. Bu yöntemde veri seti dört gruba ayrılmaktadır. İlk aşamada ağı eğitmek için iki veri grubu kullanılır. İkinci aşamada ağı test etmek için bir grup veri seti kullanılır. Son veri seti grubu ise eğitim ağının çıktı değerlerini tahmin etmek için kullanılır. Bu süreç, minimum seviyede hata oranı elde edilinceye kadar farklı sayıda nöronlarla tekrarlanır.

- **Sıralı Ortogonal Yaklaşımı** (Berry ve Linoff, 1997)

Gizli nöronun belirlenmesi için başka bir yaklaşım, sıralı ortogonal yaklaşımdır. Bir nöron eklenirken, bu nöron tarafından tanıtılan yeni bilgi, daha önce eklenmiş gizli nöronların çıktı vektörleri tarafından sürece alınır. Hata oranı yeterince düşük olana kadar gizli nöronların sayısı tek tek artırılır. Bu yöntemin ek avantajı, yapay sinir ağlarını karışık tipte gizli nöronlarla kurup eğitmek ve böylece hibrit modeller geliştirebilmektir.

3.3.3. Veri seti büyüklüğü

Modeli eğitirken veri setinin büyük ve çeşitli olması, ağın öğrenmesini pozitif yönde etkileyecektir. Ancak veri seti hacmi büyüdükçe öğrenme performansı da aynı ölçüde artmaz. Bir noktadan sonra azalarak artmaya başlayacaktır.

Buna karşılık veri hacmi büyüdükçe modeli eğitme maliyeti ve zamanı da artmaktadır. Bu bakımdan benzer başarımın elde edilebileceği en küçük veri setine karşılık gelen kırılım noktasını bulmak önem taşımaktadır.

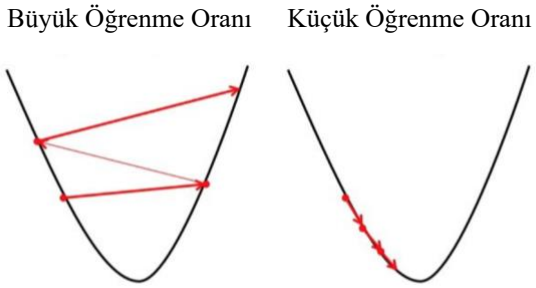
3.3.4. Öğrenme iterasyonlarının sayısı:

İterasyon sayısını belirlemek bir yapay sinir ağı modeli için önem taşımaktadır. Öğrenme süreci, aykırı bir gelişme tespit edilince durdurulmalıdır. Aksi takdirde aşırı uyum meydana gelebilir. Örneğin, doğrulama hatası ile test hatasının arasındaki fark artmaya başladığında eğitim hala devam ederse aşırı uyum sorunu meydana gelir. Aşırı uyum, ağın performansını ve

tahmin yeteneğini kötüleştirir. En uygun iterasyon sayısını belirlemek için deneme yanılma yöntemi kullanılabilir (Savic, 2009).

3.3.5. Öğrenme Oranı:

Ağırlıkların istenilen sonuca ulaşana kadar sürekli yenilenmesiyle geçen zamana öğrenme denir. Öğrenme oranları, bağlantıların ağırlık değerlerindeki değişim miktarını belirleyerek ağın performansını önemli ölçüde etkiler. Ağırlıkların yetersiz çözümler arasında salınımlarına yol açmayacak olası en büyük öğrenme oranı kullanılmalıdır.



Şekil 3-7 Öğrenme Oranı ve Gradyan iniş

Öğrenme oranı küçük değerler aldığında, öğrenme işlemi uzun süreler alır ve aşırı uyum meydana gelebilir. Öğrenme oranı büyük değerler aldığında ise, ağırlıklardaki değişim de daha büyük olacağından ağ daha hızlı öğrenir ancak yanlış noktalara yakınsama olabilir. Şekil 3.7'de (www.towardsdatascience.com) büyük ve küçük öğrenme adımlarıyla Gradyan iniş gösterimi yer almaktadır. Bu nedenle yakınsama veya ayrılma öğrenme oranlarını bulmak için Grid arama yapılabilmektedir. Öğrenme oranı [0, 1] aralığında seçilen reel bir sayıdır.

Öğrenme süreci esnasında net girdi değerleri hesaplanırken en önemli adımlardan biri ağırlıkların güncellenmesi olmaktadır. Öğrenme oranının ve detaylarına bir sonraki bölümde yer verilecek olan momentum değerinin modele etkisinin daha iyi anlaşılabilmesi için, ağın öğrenmesi esnasında ağırlıklarının nasıl güncellendiğini gösteren formül aşağıda yer almaktadır.

$$\Delta A_{jm}^{a}(t) = \lambda \delta_m Ç_j^a + \alpha \Delta A_{jm}^{a}(t-1) \qquad (3.6)$$

Örneğin hata geriye yayılırken çıktı katmanı ile ara katman arasındaki bağlantı ağırlıklarının değişimini ele alalım. Formül 3.6.'da, ara katmanda bulunan j. işlem elemanını, çıktı

katmanındaki m. işlem elemanına bağlayan ağırlık değerindeki değişim miktarı ΔA^a ile gösterildiğinde, t. iterasyondaki ağırlık değişimi gösterilmiştir. Eşitlikteki λ öğrenme katsayısını, α momentum katsayısını, A ağırlık değeri, λ öğrenme oranı, α momentum oranı, Ç çıktı değeri, δ hata oranını göstermektedir. Öğrenme esnasında eşik değeri güncellemesi de oldukça benzer bir süreçten geçtiği için, öğrenme oranının eşik değerindeki değişime etkisi de benzerdir (Olgun, 2009).

3.3.6. Momentum:

Momentum ve öğrenme oranı yakından ilişkilidir. Optimal öğrenme oranı momentuma, momentum öğrenme oranına bağlıdır. Öğrenme oranı, ayarlanması gereken en önemli hiperparametre olarak kabul edildiğinden, momentum da önemlidir. Bunun yanı sıra bütün YSA' larda öğrenme katsayısı vardır fakat momentum katsayısı yoktur.

Momentum, ağın öğrenme sırasında daha hızlı toparlanmasına ve önceki adımı hatırlamasına yardım eden bir faktördür. Daha kısa zamanda minimuma yakınsamak için bir önceki iterasyondaki değişimin belirli bir oranda yeni değişim miktarına eklenmesi şeklinde ifade edilebilir. Böylece önceki adımların bilgisi ile bir sonraki adımın yönünü bilmeye yardımcı olur.

0 ile 1 arasında değerler alır. Tipik bir momentum seçimi 0,5 ile 0,9 arasındadır. Öğrenme oranları gibi eğitim sırasında kararsızlığa neden olmadan momentumu mümkün olduğunca büyük ayarlamak etkilidir.

3.3.7. Mini parti büyüklüğü(Mini batch size):

Mini parti sayısı, modelin aynı anda kaç veriyi işleyeceği anlamına gelmektedir. Diğer bir deyişle ağa verilen alt örneklerin sayısıdır.

Daha büyük bir parti ile çalışmak daha büyük bir öğrenme oranı kullanılabilmesini sağlamaktadır. Ancak büyüklük genellikle CPU belleği tarafından sınırlandırılır (Smith, 2018) Buna çözüm olarak model eğitilirken veriler belli sayıda parçalar(mini-batch) halinde eğitimde yer alabilirler. İlk parça eğitilip modelin başarımı test edildikten sonra, yeni eğitim kümesi ile model tekrar eğitilip, ağırlıklar tekrar güncellenir. Bu işlem tekrarlanarak model için en uygun ağırlık değerleri hesaplanmaya çalışılır.

Mini parti değeri 1 ile eğitim kümesindeki tüm verilerin sayısı aralığında bir değer olarak belirlenebilir. Mini parti değeri, alabileceği en küçük değer olan 1 olarak belirlendiğinde yani her iterasyonda sadece tek bir veri üzerinde işlem yapılırsa, süreç stokastik gradyan iniş ile aynı olacaktır. Tüm data tek seferde verilirse de parti gradyan iniş yapılmış olunur ancak yüksek maliyete neden olur. Bu nedenle ideal parti büyüklüğünü belirlemek önem taşımaktadır. CPU bakımından makine öğrenme alanında yaygın olarak kullanılan mini parti büyüklükleri 32, 64, 128 ve 256 olmaktadır (Wong ve diğ, 2018).

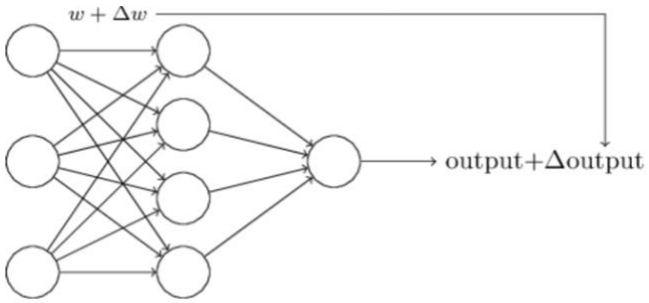
Yapılan bazı çalışmalarda verilerin bu şekilde partiler halinde işlenmesi durumunda kayıp değerinin arttığı fakat zamandan kazanıldığı tespit edilmiştir. Ayrıca veri setinde fazla miktarda gürültülü veri varsa, başarım grafiği yüksek oranda inişli çıkışlı bir gidişat sergileyebilir. Bu tür durumlarda eğer veriyi değiştirme şansı yoksa, parti değeri değiştirilerek problem çözülebilir.

3.3.8. Başlangıç ağırlığı ve ağırlık değişimleri:

Başlangıç ağırlığı, bir ağın eğitim süresi ve ortaya çıkan modelin kalitesi üzerinde etkisi olan, önemli bir faktördür. Başlangıç değerleri, yerel minimumda salınma olasılıklarını değiştirerek öğrenme sürecinin zorluğunu etkileyebilir. Yanlış bir başlangıç ağırlığı, gradyan inişin yakınsamasını önleyebilir.

Ağırlık başlatma teknikleri üç ana gruba ayrılır. Bunlar, mevcut verilerden elde edilen bilgi miktarına bağlı olarak Veri Bağımsız, Veri Bağımlı ve Eğitim Öncesi yaklaşımlar olmaktadır (Aguirre ve diğ, 2019). Başlangıç ağırlığı 0 olamaz. Çünkü öyle bir durumda katmanlardaki hesaplar hep aynı çıkacağından öğrenme gerçekleşmemiş olacaktır.

Bir yapay sinir ağının öğrenme sürecinde öncelikle ağın, ağırlık ve sapmaları öğrenmesi istenir. Böylece ağdan gelen çıktı, girdileri doğru bir şekilde sınıflandırabilir. Ağdaki bazı ağırlıklarda (veya sapmalarda) küçük değişiklikler yapılırsa, bu değişikliklerin ağdan çıkışta sadece küçük bir değişime neden olması beklenir. Bu özellik öğrenmeyi mümkün kılacaktır (bkz Şekil 3.8).



Şekil 3-8 Öğrenme sürecinde ağırlık değişimlerinin gösterimi (Nielsen, 2015)

Eşik değeri, sinir hücrelerinin gelen uyarıya tepki vermesi için geçilmesi gereken bir koşul değeridir. İnsan beyninde ancak eşik değerinin aşılması koşuluyla sinyaller işlenip bir sonraki sinir hücresine aktarılmaktadır. YSA'da da gelen sinyal, sinir hücresinin eşik değerini aşmıyorsa sonraki sinir hücresine iletilmez. Formül 3.1'de de görüldüğü gibi net girdi değeri, ağırlıklarla çarpılmış girdi değerlerinin eşik değeri ile toplanması sonucu elde edilir.

Eşik değeri, ağın çıktısının 0 olmasını önler, öğrenmeyi hızlandırır ve yerel minimuma takılma riskini azaltır.

3.3.9. Kayıp Fonksiyonları

Kayıp fonksiyonları makine öğrenmesinde ağın hatasını ölçen hiperparametrelerdir. Derin ağların son katmanı, kayıp fonksiyonunun tanımlandığı katmandır. İyi bir modelden beklenti, sıfıra yakın bir kayıp fonksiyonuna ulaşmaktır.

Kayıp fonksiyonu hata hesaplama sürecini bir optimizasyon problemine dönüştürerek yaptığı için bu fonksiyon amaç ya da maliyet fonksiyonu olarak da anılmaktadır. Kayıp değeri hesaplayan bir çok fonksiyon türü mevcuttur. Bunlara örnek olarak Hinge, Huber, Kullback Leibler, MAE (Ortalama Mutlak Hata), MSE (Ortalama Hata Karesi) ve Cross Entropy verilebilir. En sık kullanılan kayıp fonksiyonlarının açıklamaları aşağıda yer almaktadır.

- **L1 ve L2 Kayıp Fonksiyonları**

L1 kayıp fonksiyonu ortalama mutlak hata, L2 kayıp fonksiyonu ortalama hata karesi olarak da anılmaktadır. Formülleri aşağıda yer almaktadır.

$$\text{L1 kayıp fonksiyonu} = \text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|y_{gerçekleşen} - y_{beklenen}\right| \quad (3.6)$$

L2 kayıp fonksiyonu = MSE = $\frac{1}{n}\sum_{i=1}^{n}\left(y_{gerçekleşen} - y_{beklenen}\right)^2$ (3.7)

L1 kayıp fonksiyonu, gerçek değer ile beklenen değer(tahmin) arasındaki farkın mutlak değerlerinin toplamı olmaktayken, L2 kayıp fonksiyonu bu farkın karelerinin toplamıdır.

- **Çapraz Entropi Kaybı**

Çapraz Entropi(Cross Entropy), iki olasılık dağılımı arasındaki farkı belirsizlik temelinde hesaplayan bir kayıp fonksiyonudur. Formül 3.8'de görüldüğü gibi q tahmin edilen değerin, p gerçek değerden ne kadar değişiklik gösterdiği dikkate alınır.

$$H(p,q) = -y \log \hat{y} - (1-y)\log(1-\hat{y}) \quad (3.8)$$

Tahmin edilen olasılık gerçek değerden uzaklaştıkça çapraz entropi kaybı artar; yaklaştıkça da azalır. İdeal modele ulaşmak için bu değer azaltılmaya çalışılır.

3.3.10. Regülasyon(Regülarizasyon)

Regülasyon teknikleri ağı aşırı uyuma karşı koruyarak karmaşıklığı azaltma amacıyla uygulanan iyileştirme teknikleridir. Literatürde Erken durdurma, Seyreltme, Öğrenme oranını azaltma, Parti normalizasyonu, Veri seti boyutunu artırma gibi regülasyon tekniklerine rastlanmaktadır. Bu bölümde sık kullanılan bazı regülasyon teknikleri anlatılacaktır.

- **L1 ve L2 regülarizasyonu:**

En sık uygulanan regülasyon teknikleri L1(Lasso) ve L2(Ridge) teknikleridir. L1 ve L2 regülasyonu, ağırlığı yüksek olan değişkenlerin daha küçük değerlerde kalmasını sağlayacak şekilde ağı zorlayarak, bu değişkenlerin etki oranını azaltır. L1 ve L2 kayıp fonksiyonuna eklenen bağımsız bir parametredir. Zorunlu olmamakla beraber kullanılması önerilmektedir.

Regülasyon işlemi doğrudan fonksiyonda değişikliği sağlamaktadır ve sadece ağırlık değerleri etkilenmektedir. Başlangıç değerine yönelik bir regülasyon yapılmamaktadır. Değişkenlerin ağırlığını azaltmak için regülasyon değerini artırmak gerekmektedir. Eklenen bu parametre sayesinde kayıp fonksiyonu çıktısı hiçbir zaman sıfır olmayacaktır. Kayıp fonksiyonunun çıktısının sıfır olabilmesi için ancak tüm ağırlıkların sıfır olması gerekmektedir. Bu yönteme ağırlık azaltımı (weight decay) da denmektedir.

Ağırlık azaltımını, ağı aşırı uyuma ya da gürültülü girdi verilerine karşı daha sağlam hale getirmek için, kayıp fonksiyonuna eklenen bir ceza terimi olarak düşünebiliriz. Bu ceza terimi hiperparametre ile parametre vektörü normunun çarpıdır. Ceza süresinin artırılması (ağırlık azalımı hiperparametresinin arttırılması), genelleştirmeyi geliştirir. Bu yöntemle, bir yapay sinir ağında girdilerin her birinden gelen ağırlıklar için farklı bir ağırlık bozulması gerçekleşir. Girdilerden birinde büyük bir ağırlık bozulması olması, ilgili diğer ağırlıkları da etkili bir şekilde çok küçük değerlere zorlar (Bengio, 2000).

- **Seyreltme(Dropout):**

Seyreltme yöntemi sık kullanılan regülasyon yöntemlerinden biridir. Ağın, öznitelikleri ağ boyunca daha eşit bir şekilde temsil edebilmesi için, bazı nöronlara bazı girdi bağlantıları devre dışı bırakılır. Diğer bir anlatımla, belli kurallara göre ya da rasgele olacak şekilde girdi katmanı ve gizli katmandan belli nöronların kaldırılmasıdır.

Seyreltme eğitim tekniği ile nöronların bir kısmı her eğitim dönemi için sıfıra ayarlanır. Böylece katmanlardaki ilgili bağlar koparılmış olunur. Nöronlar birbiri hakkında daha az bilgiye sahip olur ve nöronlar birbirlerinin ağırlık değişimlerinden daha az etkilenirler. Bu sayede daha tutarlı modeller oluşturulabilir.

Bu teknik, gizli katman nöronlarının birlikte adaptasyonunu önleyerek YSA'ların genelleştirme yeteneğini arttırmaktadır. Her bir katmanda farklı gizli birim kombinasyonları birbiriyle çalıştığı için daha iyi bir öğrenme gerçekleşecektir. Bu eğitim özelliğiyle, her gizli nöron, öznitelik alanından çıktı alanına kadar bağımsız bir eşleşme oluşturmalıdır. Böylece, gizli katmandaki seyreltme, YSA'ların girdi verilerini ezberlemesini önlemeye yardımcı olur (Mendenhall ve diğ, 2016).

- **Parti normalizasyonu:**

Parti normalizasyonu(batch normalization), eğitim sürecini hızlandırmak için kullanılan bir regülasyon yöntemidir. Yöntem, yapay sinir ağındaki her katmana girişleri normalleştirir. Bu işlem, gizli bir katmandaki ağırlıkların ne kadar değişeceğini sınırlayarak yapılır. Alt katmanlardaki değişimler çok büyükse, üst katmanlardaki nöronlar sadece iyi tahminlerin nasıl yapılacağını öğrenmek zorunda kalmaz, aynı zamanda önceki katmanlardaki ağırlık

değişimlerini de karşılar. Böylece, eğitim süreci yavaşlayacak, yapay sinir ağının ağırlıkları hızlı bir şekilde çözüme yakınsamayacaktır (Buduma ve diğ, 2017).

Normalize edilmeyen veri seti ile bir ağı eğitmek, hem ağın eğitim süresini uzatacak hem de ağın verimini düşürecektir. Uygulanan birçok normalizasyon çeşidi bulunmaktadır. Bunlara örnek olarak Min-Max kuralı, Z-skoru, Binning verilebilir (Hovden, 2019).

- ***Binning normalizasyon:***

Öznitelik değerlerinden eşit büyüklükte gruplar oluşturulur. Her gruptaki her değer, toplam grup sayısına bölünerek ilgili özniteliğin normalleştirilmiş değeri bulunur.

- ***Min-Max normalizasyon:***

En sık kullanılan normalizasyon yöntemlerinden biri Min-Max yöntemidir. Bu yönteme göre her öznitelik, aşağıdaki denklem kullanılarak [0,1] aralığında yeniden ölçeklendirilir.

$$v' = \frac{v - min_A}{max_A - min_A} \tag{3.9}$$

Burada, min_Ave max_A, A özniteliğinin minimum ve maksimum değerleridir. A özniteliğinin orijinal ve normalize edilmiş değeri sırasıyla v ve v' ile temsil edilir. Minimum değerin orijinal değerden farkı, maximum ve minimum değerin farkına bölünerek normalize edilmiş değer bulunur. Yukarıdaki denklemden, maksimum ve minimum öznitelik değerlerinin sırasıyla 1 ve 0'a eşlendiği görülmektedir.

- ***Z-skoru Normalizasyonu:***

Z-skoru normalizasyonu için özniteliğin normalleştirilmiş değeri aşağıdaki denklem kullanılarak hesaplanabilir.

$$z = \frac{v - \mu_A}{\sigma_A} \tag{3.10}$$

Burada μ_A ve σ_A, A özniteliğinin ortalaması ve standart sapmasıdır. Orijinal ve normalleştirilmiş öznitelik değerleri, sırasıyla v ve z ile temsil edilir. A özniteliğinin ortalamasının orijinal

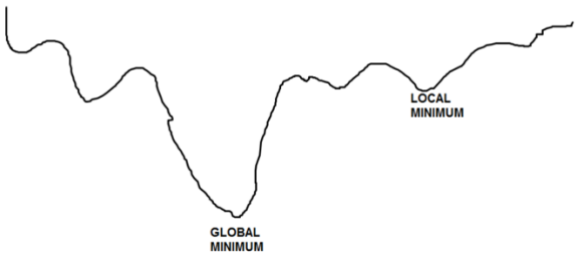
değerden farkı, A özniteliğinin standart sapmasına bölünerek normalize edilmiş değer bulunur. Normalleştirmeden sonra, tüm öznitelik değerlerinin ortalaması ve standart sapması sırasıyla 0 ve 1 olur (Jain ve diğ, 2018).

3.4. DSA'DA HİPERPARAMETRE SEÇİMİ İÇİN DİKKAT EDİLMESİ GEREKEN DURUMLAR

Bir yapay sinir ağının optimum çözüme en hızlı yoldan ulaşabilecek şekilde modellenebilmesi için kullanılan hiperparametreler oldukça büyük önem taşımaktadır. Yanlış bir modelleme olması durumunda hatalı sonuçlar elde edilebilmektedir. Bu gibi örneklerle ne tür durumlarda karşılaşılabileceği, nelere dikkat edilmesi gerektiği ve bu durumlar karşısında uygulanabilecek çözüm yöntemleri bu bölümde anlatılmaktadır.

3.4.1. Ağın Yerel Minimumda Salınımı

Yapay sinir ağının optimum çözüm yerine yetersiz bir çözümü en iyi çözüm zannederek orada takılıp kalması anlamına gelmektedir. Diğer bir deyişle, ağın global minimum yerine yerel minimumu çözüm zannederek orada salınmasıdır. Şekil 3.9'de (www.towardsdatascience.com) örnek bir salınım gösterimi yer almaktadır.



Şekil 3-9 Ağın yerel minimumda salınımı

Gizli katmanın nöronları, istenen çıktıya yaklaşan bir çıktı üretmeden önce doyum noktalarına (0 ve 1) yaklaşan bir çıktı ürettiğinde, ağ yerel minimumda tuzağa düşebilmektedir. Bu gibi durumlar sonrasında artık ağırlıklar güncellenmez, hata sinyali geri yayılmaz, girdi sinyali de ileri doğru akmaz.

Ağırlık değerlerinin başlangıçta çok büyük verilmesi ya da öğrenme oranının çok yüksek belirlenmesi bu duruma sebep olabilmektedir.

Öğrenme algoritmasını yerel minimumdan kaçacak şekilde, önceki yönde tutması için momentum kullanılır. Öğrenme algoritmalarının amacı yerel minimum noktasından kurtarıp daha iyi olan global minimum noktasını bulmaktır.

3.4.2. Aşırı Uyum(Overfitting)

Ağın değişkenler arası gerçek ilişkiyi öğrenmek yerine gürültüyü öğrenmesi sonucunda genelleştirme yeteneğini kaybetmesi ve ezberlemesidir.
Aşırı uyum durumunda ağ sadece eğitildiği veri setini öğrenir, daha önce karşılaşmadığı örnekler karşısında başarısız tahminler yapar.

Eğitim sırasında aşırı uyumun meydana gelmesinin bir çok nedeni olabilir. En başlıca nedenler aşağıda listelenmiştir.

- Devir sayısının olması gerekenin çok üzerinde olması
- Eğitim verisinin modelin parametre sayısına göre yetersiz miktarda olması
- Modelin, gözlem miktarına göre fazla sayıda özellik barındırması
- Eğitim verisine göre ağırlık sayısının yüksek olması
- Gizli nöron sayısının fazla olması
- Verinin yeterli çeşitlilikte olmaması
- Varyansın çok yüksek olması

Aşırı uyum durumlarında farklı çözüm yöntemleri uygulanabilir. Eğer veri setinde birbirleriyle korelasyonu yüksek veri sütunları varsa ilgili sütunlar silinebilir ya da çeşitli yöntemlerle birleştirilebilir. Veri setinin çeşitliliği düşükse daha fazla veri ekleyerek veri çeşitliliği artırılabilir. Karmaşık bir model ile çalışılıyorsa da regülasyon yöntemleri uygulanabilir. Regülasyon yöntemlerinin detayı 3.3.11 Regülasyon bölümünde anlatılmaktadır. Aşırı uyum durumlarında en çok başvurulan bir diğer çözüm yöntemi ise erken durdurmadır. Bu yöntem, aşağıda Validasyon konusu içerisinde anlatılacaktır.

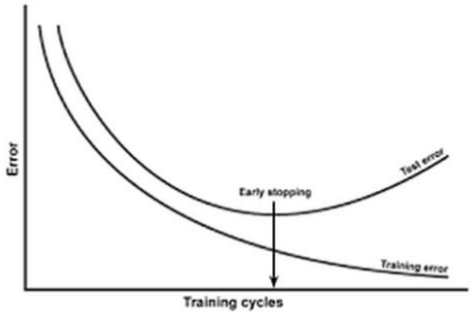
- **Validasyon:**

Modelin eğitilmiş mi yoksa ezberlemiş mi olduğu kontrol etmek için en çok uygulanan yöntem Validasyon yani Doğrulama yöntemidir. Bunun için öncelikle veri seti, Eğitim, Doğrulama ve Test verileri olmak üzere rasgele 3 alt gruba ayrılır. Şekil 3.10'da (www.towardsdatascience.com) gösterildiği gibi, test ve doğrulama veri seti eşit miktarlarda olmalıdır. Genellikle %60 Eğitim, %20 Doğrulama ve %20 Test veri seti olarak ayrılır.



Şekil 3-10 Validasyon

Test veri seti, modelin başarımını ölçen ve eğitimde kullanılmayan, tamamen bağımsız bir veri setidir. Doğrulama veri seti ise eğitim setinin bir parçası olacak şekilde eğitime doğrudan katılır. Eğitim sonunda her iki veri setinin de hata değerleri ölçülür. Şekil 3.11'de görüldüğü gibi test hatası ile eğitim hatasının arasındaki fark açılmaya başladığında ağ öğrenmiyor, ezberliyor demektir. Bu aşamada eğitim durdurulur. Bu yönteme Erken Durdurma (Early Stopping) yöntemi denmektedir. Erken Durdurma, bir regülasyon tekniğidir. Aşağıdaki şekilde de görüldüğü üzere erken durdurma yöntemi uygulanarak ağın genelleştirme yeteneğini kaybetmesi engellenebilmektedir.



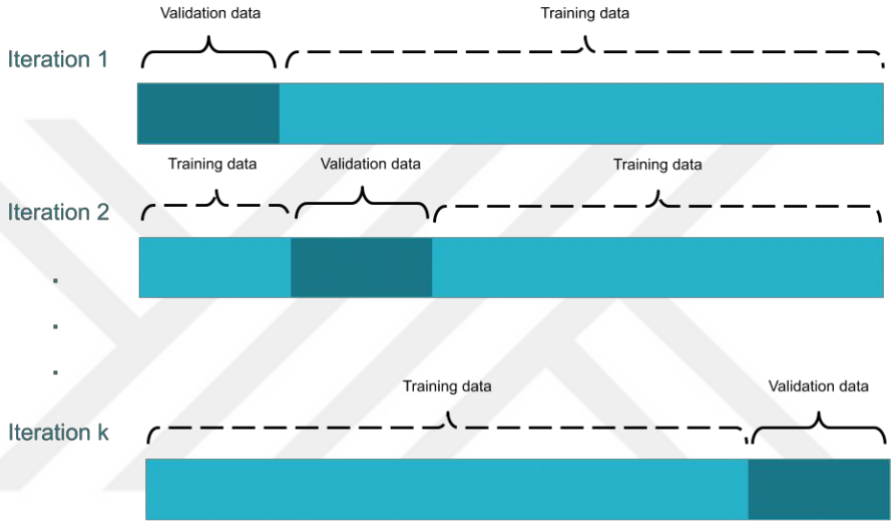
Şekil 3-11 Erken Durdurma

Bu yöntemin bazı riskleri de bulunmaktadır. Bunlardan biri, test verisi ayrılarak eğitime katılmadığı için veri yetersizliği nedeniyle yetersiz uyum meydana gelme riski, diğeri ise validasyon için ayrılan veri setinin eğitim kümesini iyi temsil edememe riskidir. Bu durumlara karşılık her iterasyonda farklı bir doğrulama veri setinin eğitime katıldığı Çapraz Doğrulama yöntemi önerilmektedir.

- **k-katlı Çapraz Doğrulama Yaklaşımı:**

Bu yaklaşımda, veri seti rastgele karıştırılır ve eşit boyutta k gruba bölünür. Doğrulama seti olarak her seferinde farklı bir kat kullanılırken, kalan katlar modeli eğitmek için kullanılır. Bu durum, Şekil 3.12'de(Almquist ve diğ, 2019) görselleştirilmiştir.

Çapraz Doğrulama hatası, k farklı validasyon hatasının ortalaması alınarak hesaplanır. Sonucun modeli eğitmek için kullanılan veri noktalarının seçiminden etkilenmemesini sağlamak için, verilerin rastgele karıştırılması gerekir.



Şekil 3-12 k-Katlı Çapraz Doğrulama yaklaşımının gösterimi

Bu yöntem sayesinde hem tüm veri seti eğitime katılır, hem de farklı örneklemler ile test edilip ortalama değer alınarak daha sağlıklı bir hata değeri elde edilir.

3.4.3. Yetersiz Uyum(Underfitting)

Eğitim veri seti üzerinde modelin olması gerekenden basit olarak kurgulanması sonucunda başarısız tahminler üretmesi ve bunun sonucunda da gerçek değerlerle modelin ürettiği tahmin değerlerinin arasındaki hata değerinin yüksek olma durumudur. Bu gibi durumlarda katman ya da nöron sayısı yetersizse daha fazla katman ve nöron eklenebilir. İterasyon sayısı yetersizse de öğrenme süreci daha uzun tutulabilir.

Optimum çözüme ulaşmadan önce eğitimin durdurulması yetersiz uyuma neden olabilmektedir. Bu bakımdan öğrenme sürecinin durdurma kriterinin doğru şekilde belirlenmesi önem taşımaktadır. Öğrenme sürecini başlıca durdurma yöntemleri şöyle sıralanabilir; belirli bir iterasyon sayısına gelince, belirli bir hata oranına ulaşınca ya da elde edilen hata değeri bir

öncekinden büyükse durdurma. Düşük bir başlangıç değeriyle başlayıp performans iyileşince manuel olarak da öğrenme süreci durdurulabilir.

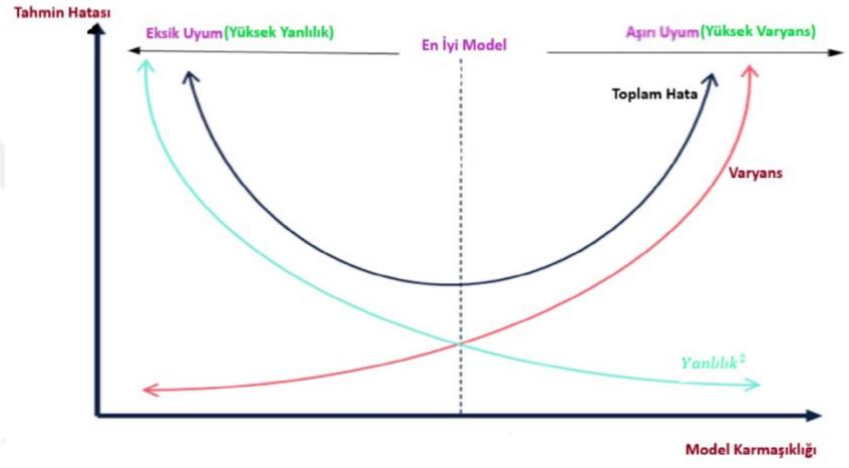
Yanlılık-Varyans-Uyum İlişkisi:

Yetersiz uyuma yol açan nedenlerden biri de yüksek yanlılıktır. Yüksek yanlılık, modelin yanlış varsayımlara dayandığını göstermektedir. Örneğin doğrusal olmayan bir veri seti, doğrusal kabul edilerek modellenirse, bu model çözüm için zayıf kalır ve yetersiz uyuma neden olur.

Yanlılık, modelin doğruluğuyla, varyans ise hassasiyetiyle ilişkilidir. Varyans ve doğruluk arasında test orantı bulunmaktadır. Düşük varyansa sahip bir model, yüksek doğruluk gösterirken, yanlılığın yüksek olması durumunda ortalama hata oranı da yüksek olur.

Yanlılığın çok düşük, varyansın çok yüksek olması durumunda aşırı uyum, Yanlılığın çok yüksek, varyansın çok düşük olması durumunda da yetersiz uyum riski söz konusudur.

Yüksek yanlılık durumlarından kaçınmak için modelin karmaşıklığı artırılabilir. Modelin karmaşıklığı arttıkça varyans artacak, yanlılık azalacaktır. Şekil 3.13'te (Yangın, 2019) görüldüğü gibi varyans ve yanlılığın kesiştiği noktada hata oranı en düşüktür yani optimal çözüm budur.



Şekil 3-13 Yanlılık - Varyans -Uyum Dengesi

Modelin karmaşıklığını artırmak için modele veri eklenerek veri çeşitliliği artırılabilir, modele yeni değişkenler eklenebilir ya da değişkenlerin ağırlığını arttırmak için regülasyon değeri azaltılabilir.

Uyum sorunu olan modellerde yetersiz uyum durumu, hata değerinin yüksek çıkması nedeniyle hemen tespit edilebileceği için, asıl dikkat edilmesi gereken aşırı uyum durumlarıdır. Çünkü aşırı uyum durumunda hata hesaplamaları esnasında model başarılı gibi düşünülerek yanıltabilir. Literatürde yetersiz uyumdan ziyade aşırı uyum örnekleriyle daha sık karşılaşılmaktadır.

3.5.GÜÇLENDİRİLMİŞ KARAR AĞACI HİPERPARAMETRELERİ

Güçlendirilmiş karar ağacı hiperparametrelerinin yüksek ya da düşük olmasının ilgili hiperparametreye göre çeşitli avantajları ve riskleri bulunmaktadır. Bu nedenle güçlendirilmiş karar ağacı modeli oluşturulurken hiperparametre ayarlama aşamasında dengeli değerler belirlenmesi önem taşımaktadır.

Güçlendirilmiş karar ağacı modellerinde en sık kullanılan hiperparametreler ve modele etkileri aşağıda açıklanmıştır.

3.5.1. Ağaç başına maksimum yaprak sayısı:

Bir ağaçta oluşturulabilecek maksimum yaprak sayısıdır. Ağaç derinliği de denebilir. Bu değerin arttırılması, potansiyel olarak ağacın boyutunu büyüterek doğruluğunu artırırken, aşırı uyum ve uzun eğitim süresi riskini de beraberinde getirir.

3.5.2. Yaprak düğümü başına minimum örnek sayısı:

Bir ağaçta herhangi bir yaprak oluşturmak için gereken minimum vaka sayısıdır. Bu değerin arttırılması, yeni kurallar oluşturma eşiğini de artırır. Örneğin, varsayılan değer 1 olduğunda, tek bir vaka yeni bir kuralın oluşturulmasına neden olabilirken, değer 5'e çıkarıldığında, yeni kural oluşturulması için eğitim verilerinin aynı koşulları karşılayan en az 5 vaka içermesi gerekir.

3.5.3. Öğrenme oranı:

Derin sinir ağlarında da kullanılan bir hiperparametredir. Öğrenme esnasındaki adım boyutunu tanımlayan 0 ile 1 arasında bir değerdir. Öğrenme oranı, uygulayıcının optimal çözüme ne kadar hızlı ya da ne kadar yavaş yaklaştığını belirler. Adım boyutu çok büyükse, en uygun çözüm aşılabilir. Adım boyutu çok küçükse de eğitimin en uygun çözüme kavuşması uzun süreler alabilir.

3.5.4. Oluşturulan ağaç sayısı:

Karar ağacı topluluğunda oluşturulacak toplam karar ağacı sayısıdır ve iterasyon sayısını belirler. Daha fazla karar ağacı oluşturularak potansiyel olarak daha iyi kapsama alanı elde edilir ve performans artar. Ancak bu durumda eğitim süresi de artar. Değer 1 olarak ayarlanırsa ilk parametre kümesine sahip tek bir ağaç görülür. Ancak bu durumda yalnızca bir ağaç üretildiği için, başka yineleme yapılmaz (Microsoft Azure Documentation, 2020).

3.6.REGRESYON DEĞERLENDİRME METRİKLERİ

Regresyon modelleri oluşturulduktan sonra değerlendirme ve karşılaştırma yapılabilecek çok sayıda ölçü birimi vardır. Bunlardan bazıları aşağıda açıklanmıştır.

3.6.1. Ortalama mutlak hata (MAE:Mean Absolute Error):

Tahminlerin gerçek sonuçlara ne kadar yakın olduğunu mutlak değer içinde ölçen bir değerlendirme ölçümüdür. Tahmin edilen değerin gerçek değerden az ya da çok olmasının bir önemi yoktur, ne kadar az ya da ne kadar çok olduğu önemlidir.

Orijinal verilerle aynı birime sahiptir. Regresyon değerlendirmeleri için en çok kullanılan metriklerden biridir.

a gerçekleşen değer, p tahmin değeri olmak üzere tahmin değerleriyle gerçek değerler arasındaki mutlak farkın ortalaması alınarak elde edilir.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|p_i - a_i| \quad (3.11)$$

3.6.2. Ortalama hata kare kökü (RMSE: Root Mean Squared Error):

Tahmin değerleriyle gerçek değerler arasındaki farkın kareler ortalamasının kare kökü alınarak elde edilir.

RMSE her zaman MAE değerine eşit veya bu değerden büyük olacaktır. Aralarındaki fark, gerçek ve tahmin değerleri arasındaki bireysel hatalarda, sapma büyüdükçe artar. Eğer RMSE ve MAE eşitse, tüm hatalar aynı büyüklüktedir.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(p_i - a_i)^2} \quad (3.12)$$

3.6.3. Göreceli mutlak hata (RAE: Relative Absolute Error):

Tahmin değerleri ve gerçek değerler arasındaki mutlak farkların toplamının, $\bar{a}$ ortalama değerlerin gerçek değerlerden mutlak farklarının toplamına bölünmesiyle elde edilir .

$$RAE = \frac{\sum_{i=1}^{n}|p_i - a_i|}{\sum_{i=1}^{n}|\bar{a} - a_i|} \quad (3.13)$$

3.6.4. Göreceli hata karesi (RSE: Relative Squared Error):

Tahmin değerlerinin gerçek değerlerden farklarının kareleri toplamının, ortalama değerlerin gerçek değerlerden farklarının kareleri toplamına bölünmesiyle elde edilir.

$$RSE = \frac{\sum_{i=1}^{n}(p_i - a_i)^2}{\sum_{i=1}^{n}(\bar{a} - a_i)^2} \quad (3.14)$$

3.6.5. Determinasyon Katsayısı r^2 (COD: Coefficient of Determination):

Bağımlı değişkendeki toplam değişimin yüzde kaçının bağımsız değişkenler tarafından belirlendiğini gösterir.

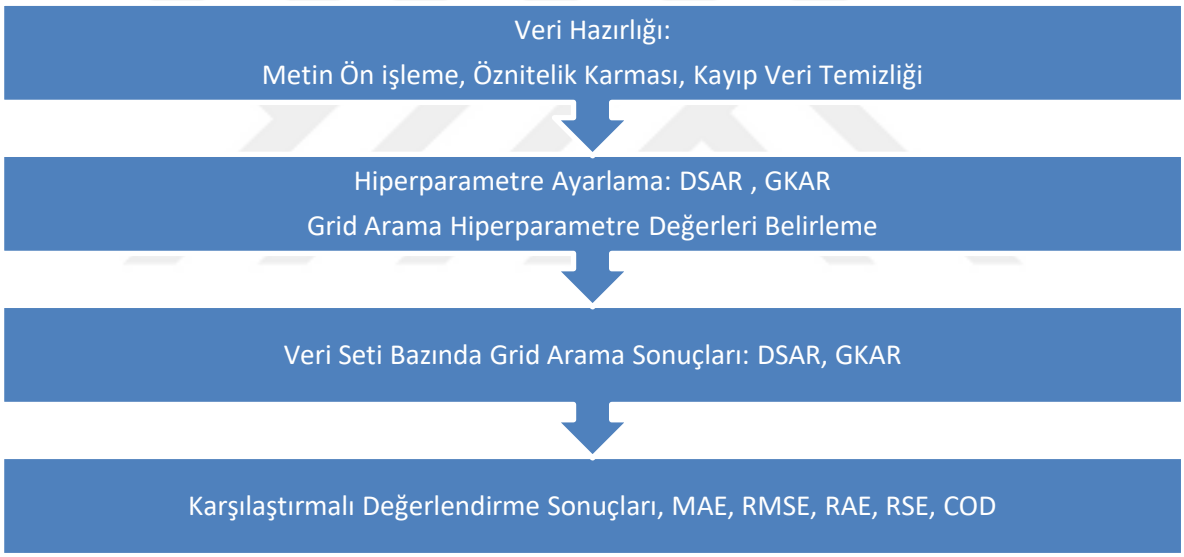
Modelin tahmin gücünü 0 ile 1 arasında bir değer olarak temsil eder. 0, modelin rastgele olduğu ve hiçbir şey açıklamadığı anlamına gelir. 1 ise mükemmel bir uyum olduğu anlamına gelir (Sayad S.).

$$r^2 = 1 - \frac{\sum(y - y')^2}{\sum(y - \bar{y}')^2} \quad (3.15)$$

4. BULGULAR

Tezin uygulama bölümünde, JIRA kullanan çevik yapıdaki 16 açık kaynaklı projenin istek verileri kullanılmıştır. Bu veriler, proje için gelen metinler halindeki istekleri ve bu isteklerin gerçekleştirilme eforunun temsil edildiği hikaye puanlarını içermektedir. Sağlıklı bir modelleme yapılabilmesi için bu veriler tahmin modellemesi süreci öncesinde bir dizi veri hazırlığı işleminden geçirilmiştir.

Uygulama sürecinde makine öğrenmesi alanında birçok algoritmayı destekleyen ve çeşitli kodlama dilleriyle yapısında özelleştirmeye müsaade eden bir bulut bilişim platformu olan Microsoft Azure Machine Learning Studio kullanılmıştır.



Şekil 4-1 Uygulama Akışı Şeması

Şekil 4.1'de de görüldüğü gibi uygulama süreci ilk olarak veri hazırlığı ile başlanmıştır. Veri setinde en çok yer kaplayan açıklama sütunu verilerinden, projeye katkı sağlamayan, sürecin ve sonucun kalitesini olumsuz etkileyebilecek unsurlar çıkarılmış ya da sadeleştirilmiştir. Sonrasında sadeleştirilen bu metin sütunu da dahil edilerek tüm metin verileri, eşit uzunluktaki sayısal vektörlere dönüştürülmüştür. Bu süreçte karma çatışmasını önleyecek kadar büyük, öğrenme sürecinde etkili olacak kadar da küçük karma boyutu ve n-gram sayısı kombinasyonunun bulunabilmesi için, örnek bir veri seti üzerinden grid arama yapılmıştır ve en düşük hata oranını veren kombinasyon seçilmiştir. Böylece karma tablosunun kapasitesi ve

uygulama şekli belirlenmiştir. Bu sürecin sonunda, metin satırlarındaki ünik kelimelerin, ilgili öznitelik karması sütununda içerilip içerilmediğine göre ağırlıklarının tamsayı halinde yer aldığı bir karma tablosu oluşturulmuştur.

Veri seti tahmin modelleme sürecinden önce yapılan son işlem olarak, eksik hikaye puanı olan satırlar tamamen silinmiştir. Veri setlerinde az miktarda eksik veri içeren satırlar olduğu için silinen bu satırların sonuç üzerindeki etkisi de oldukça düşüktür. Bu nedenle eksik veriler üzerine satırı tamamen silmek dışında başka bir işlem yapılmamıştır. Bu işlemin de tamamlanmasıyla, veriler tahmin modelleme sürecine hazır hale getirilmiştir.

Tahmin modelleme sürecinde Derin Sinir Ağı Regresyonu ve Güçlendirilmiş Karar Ağacı Regresyonu modelleri karşılaştırılarak en uygun model belirlenmeye çalışılmıştır. Her iki model için de ayrı temel hiperparametreler bulunmaktadır. İlgili hiperparametreler üzerine, her iki model için de veri seti bazında 8'er farklı kombinasyon için grid aramalar yapılmış ve en düşük hata oranlarını veren kombinasyonlar belirlenerek ilgili veri setine uygulanmıştır. Veri seti, her bir grid arama için çapraz doğrulama yöntemi ile 10 farklı parçaya(kata) ayrılıp, her bir iterasyonda diğer 9 parça eğitim verisi olarak kullanılırken, ayrılan parça test edilmiştir. Gerçekleştirilen bu 10 farklı iterasyonun sonuçlarının ortalaması, ilgili grid aramanın ortalama hata değeri olarak alınmış; en düşük hata değerini veren grid arama kombinasyonu, ilgili veri setinin hiperparametre değerleri olarak kullanılmıştır. Hata değerleri karşılaştırılırken literatürde en çok kullanılan değerlendirme metriklerinden biri olan MAE(Ortalama Mutlak Hata) dikkate alınmıştır.

Sonuç olarak her bir veri seti için Derin Sinir Ağı Regresyon modelinin hata değerleriyle Güçlendirilmiş Karar Ağacı Regresyon modelinin hata değerleri karşılaştırılmıştır ve Derin Sinir Ağı Regresyon modelinin tüm veri setlerinde daha düşük hata değerleriyle tahmin gerçekleştirdiği gözlenmiştir.

4.1.VERİ SETİ

Açık veri kümeleri son yıllarda yazılım eforu tahmininde birçok araştırma projesi için değerli hale gelmiştir. PROMISE veri deposundaki China, Desharnais, Finnish, Maxwell, ve Miyazaki veri kümeleri buna örnek olarak verilebilir.

JIRA çevik yapıları destekleyen en yaygın istek takip sistemlerinden biridir. Bu nedenle tezde JIRA kullanan veri havuzları seçilmiştir.

Tezde kullanılan veri seti, 16 açık kaynaklı projeden Apache, Appcelerator, DuraSpace, Atlassian, Moodle, Lsstcorp, Mulesoft, Spring ve Talendforge olmak üzere 9 havuzda toplanan 23.313 isteği içermektedir.

Seçilen veri havuzlarının içerikleri aşağıdaki şekildedir.

- Apache, Apache Yazılım Vakfı tarafından desteklenen projeleri içermektedir.
- Appcelerator, mobil uygulama geliştirmeye odaklanan bir dizi açık kaynak projesinden oluşmaktadır.
- DuraSpace dijital varlık yönetimi projeleri içermektedir.
- Atlassian deposunun proje yönetim sistemleri ve işbirliği araçları sağlayan bir dizi projesi vardır.
- Moodle, proje üyelerinin topluluğa kullanıcı, geliştirici, test kullanıcısı ve kalite güvence uzmanı gibi çeşitli rollerde katılmasını sağlayan bir e-öğrenme platformudur.
- Lsstcorp'un Büyük Sinoptik Araştırma Teleskopu ile ilgili araştırmaları destekleyen bir dizi projesi vardır.
- MuleSoft, Mule Studio gibi yazılım geliştirme araçları ve platform işbirliği araçları sağlar.
- Spring, uygulama geliştirme çerçevelerini destekleyen bir dizi projeye sahiptir.
- Talendforge, veri entegrasyonu ve ana veri yönetimi gibi veri yönetimi çözümleri için açık kaynaklı entegrasyon yazılımı sağlayıcısıdır.

Tezde kullanılan veri seti elde edilirken hikaye puanı 0 olan, negatif olan ya da gerçekçi olmayacak kadar büyük olan(100'den büyük) istekler filtrelenmiştir ve 23.313 istekten oluşan veri seti bu şekilde elde edilmiştir.

Bu 16 proje, hem uygulama etki alanları, hem de projenin özellikleri açısından çeşitlilik göstermektedir. Her birinin istek sayısı 352 ile 4.667 arasında değişmekle birlikte, boyutları 88 KLOC (Kilos of Lines Of Code: bin kod satırı) - 18 milyon LOC (Lines Of Code: kod satırı) arasında yer almaktadır. (bkz Tablo 4.1). Projelerde teknik açıdan farklı uygulama etki alanları

ve farklı programlama dilleri kullanılmaktadır. Bunun yanısıra farklı bölgelerden farklı katılımcıların yer aldığı ekip yapıları söz konusudur (Choetkiertikul ve diğ, 2019).

Tablo 4-1 Veri seti istek sayısı tablosu

Veri havuzu	**Proje**	**İstek sayısı**
Apache	Mesos	1.680
	Usergrid	482
Appcelerator	Appcelerator Studio	2.919
	Aptana Studio	829
	Titanium SDK/CLI	2.251
DuraSpace	DuraCloud	666
Atlassian	Bamboo	521
Moodle	Clover	384
Lsstcorp	JIRA Software	352
Mulesoft	Moodle	1.166
	Data Management	4.667
	Mule	889
	Mule Studio	732
Spring	Spring XD	3.526
Talendforge	Talend Data Quality	1.381
	Talend ESB	868
TOPLAM		**23.313**

Her bir projenin veri seti sırasıyla istek kodu, başlık, açıklama ve hikaye puanı olmak üzere 4 sütundan oluşmaktadır.

Tablo 4-2 Veri Seti Örneği

issuekey	title	description	storypoint
APSTUD-3707	Add Copy URL actions to right-click context menu of Remote view for S3 files	I was able to connect to our Appcelerator S3 bucket to drag and drog copy image files that I wanted to refer to remotely from the Desktop packaging release links webpage. But I had no easy way to tell what the various URLs were that I could use to refer to the resulting S3 objects/files. Cyberduck lets you right click and choose Copy URL > HTTPS, HTTPS, Signed URL (expiring in one hour, one day, or one week + one hour), Torrent URL.	5
APSTUD-99	Content assist popup is shown outside the screen boundaries	As you can see from the image, the JavaScript content assist popup displays outside the screen boundary to the right.	8
APSTUD-190	PHP Autocompletion for parent methods	When you overwrite a parent method you often need to call it from the new method, example: {code} <?php class Parent { public function doSomethingHard() { } } class Child { public function doSomethingHard() { $this->bla = 'foo'; return parent:: # content assist here should have doSomethingHard() } } ?> {code} Aptana does not autocomplete the methods after "parent::" Suggested fix: # Attempt to replicate # Check to see if the item is fixed in 5.4, or in a newer version of the 5.3 parsers # If not in our code and can't be fixed, link to PDT issue referencing the problem	13

Tablo 4.2'de Aptana veri setinden 3 istek örneği gösterilmektedir. İstek kodunun öğrenme sürecine katkısı olmayacağı için çalışmada ilk sütun kullanılmamıştır. Başlık ve Açıklama

sütunu metin verilerinden, hikaye puanı sütunu ise tam sayılardan oluşmaktadır. Açıklama sütununda Tablo 4.2'deki örneklerden çok daha uzun metinler de olduğu için bu sütun biz dizi metin ön işleme sürecinden geçirilerek sadeleştirilmiştir. Veri hazırlığı bölümünde detaylar anlatılacaktır.

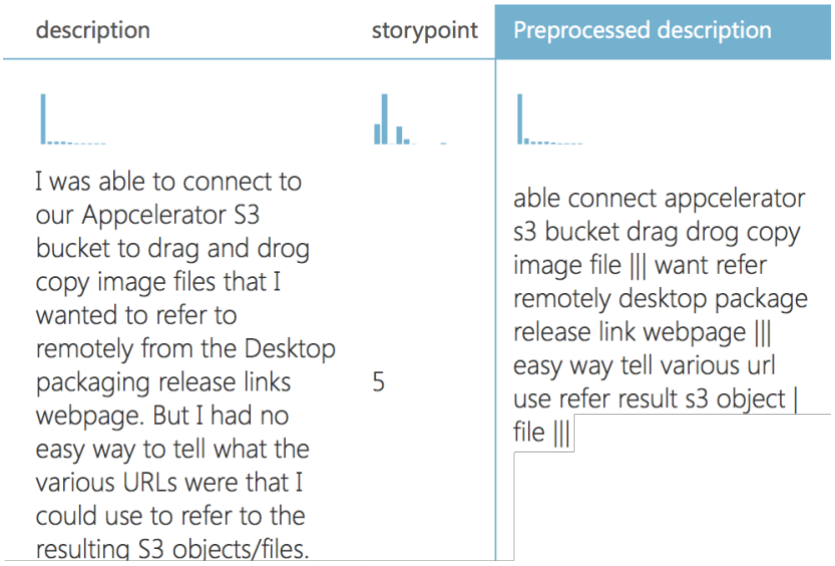
4.2.VERİ HAZIRLIĞI

Tezde kullanılan veri seti, eğitim sürecine girmeden önce metin verileri ve kayıp veriler bir dizi ön işlemeden geçirilmiştir. Bu bölümde veri setinin eğitim sürecinden önce geçtiği işlemler anlatılacaktır.

4.2.1. Metin Ön İşleme

Veri seti içerisindeki her bir projenin açıklama sütunu uzun metinlerden oluşmakla birlikte uygulamanın başarısını etkileyebilecek metin formları, işaretler ve diziler içermektedir. Öğrenme sürecinin hızını ve etkisini artırmak amacıyla bu veri sütunu sadeleştirildikten sonra sürece alınmıştır. Yapılan sadeleştirmeler aşağıdaki listelenmiştir.

- Uygulamada kullanılan veri seti İngilizce olduğu için işlem dili de İngilizce olarak seçilmiştir.
- a, the, for gibi makine öğrenmesi için bilgi içermeyen kelimeler çıkarılmıştır.
- Kelimeler kök halinde kullanılacak şekilde düzenleme yapılmıştır.
- Cümlelerin sonuna ||| sınır işareti konarak, cümlelerin tespit edilmesi sağlanmıştır.
- Büyük harfli karakterler, küçük harflere dönüştürülmüştür.
- Metin veri sütunu içerisinde yer alan dosya numarası, mesaj saati gibi makine öğrenmesine katkısı olmayan tüm sayısal karakterler kaldırılmıştır. Sadece sözcüğün bir parçası halindeki sayılar kaldırılmamıştır.
- Noktalama işaretleri gibi harf ya da rakam olmayan karakterler yerine | işareti konmuştur.
- aaaa gibi tekrarlayan diziler kaldırılmıştır:
- <dizi> @ <dizi> biçimindeki e-mail adresi içeren diziler kaldırılmıştır.
- http, https, ftp, www gibi URL öneklerini içeren diziler kaldırılmıştır.
- can't won't gibi daraltılmış fillller genişletilmiştir.



Şekil 4-2 Microsoft Azure ML, Metin ön işleme

Şekil 4.2'de uygulamada metin ön işleme sürecinden geçmiş bir açıklama metni, örnek olması bakımından gösterilmektedir. Şekilde de görüldüğü üzere özne, bağlaç gibi bilgi içermeyen kelimeler otomatik olarak kaldırılmış, kelimeler kökleri halinde kullanılmış, noktalama işaretleri dönüştürülmüş, tüm büyük harfler küçük harf olacak şekilde normalize edilmiş ve cümle sonlarına ayırıcılar konulmuştur. Yukarıda listelenen işlemlerin tüm veri setindeki açıklama metinlerine uygulanması sonucunda, ilgili uzun metinler makine öğrenmesinde daha verimli şekilde kullanılmaya hazır hale getirilerek işlem süreleri hızlandırılmış ve öğrenme sürecinin kalitesi artırılmıştır.

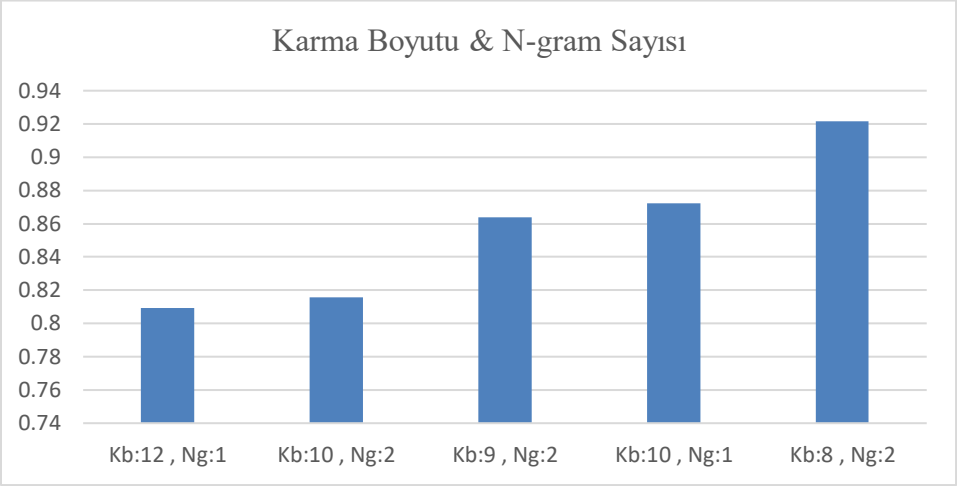
4.2.2. Öznitelik Karması

Öznitelik Karması(Feature Hashing) yönteminin uygulanacağı sütunların metin biçiminde olması gerekmektedir. Bu nedenle uygulamada başlık sütunu ve metin ön işlemeden geçmiş açıklama sütunu, öznitelik karması sürecine alınmıştır. Metin verileri öznitelik karması yöntemiyle sayısal vektörlere dönüştürülmüştür.

Öznitelik karması yaklaşımı, Vowpal Wabbit kütüphanesini uygulayan Microsoft Azure Machine Learning platformu kullanılarak gerçekleştirilmiştir. Karma fonksiyonu olarak, Vowpal Wabbit çevrimiçi öğrenme kütüphanesinde bulunan 32 bit murmurHash3 kullanılmıştır (Mehdiyev ve diğ, 2017).

Veri setine en uygun modeli bulabilmek için örnek veri seti olarak Aptana Studio seçilerek birbirine yakın ve literatürde en çok kullanılan karma boyutu ve n-gram sayıları ile determinasyon katsayısı üzerinden grid arama yapılmıştır.

Determinasyon katsayısı, regresyon modelleri için tahmin performansını temsil eden bir metriktir. 0 ile 1 arasında pozitif bir değer olup, 1'e yaklaştıkça model üzerinden gerçekleştirilecek tahminlerin doğruluk oranı da yükselmektedir. Şekil 4.3'de de görüldüğü gibi, yapılan grid aramada karma boyutunun 8, n-gram sayısının 2 olduğu kombinasyon denendiğinde determinasyon katsayısı aniden yüksek bir artış gösterdiği ve 1'e oldukça yaklaştığı için tezde bu değerler kullanılmıştır. Sonuçlar literatürdeki örneklerle tutarlı olduğu için başka bir örnek veri seti üzerine daha aynı süreç uygulanmamıştır.



Şekil 4-3 Karma Boyutu ve N-gram sayısı - Determinasyon Katsayısı Grafiği

Bit sayısı 8 olan bir karma tablosunda her bir özniteliğin giriş sayısı $2^8 = 256$ olmaktadır. Tablo 4.3'de örnek olması bakımından Aptana veri setinin karma tablosunun bir gösterimi bulunmaktadır. Sütun sayısı fazla olduğu için aralardaki bazı sütunlar gizlenmiştir.

Tablo 4-3 Karma Tablosu Örnek Gösterim

title	storypoint	Preprocessed description	title_HashingFeature_1	title_HashingFeature_12	title_HashingFeature_256	Preprocessed description_HashingFeature_1	Preprocessed description_HashingFeature_36	Preprocessed description_HashingFeature_256
Add Copy URL actions	5	able connect app	0	1	0	1	1	2
Aptana tries to open a	8	html | div | p | a	0	0	0	0	1	0
Content assist popup i	8	you image ||| ja	0	0	0	0	0	1

Tablo 4.3'de de görüldüğü gibi her bir öznitelik için 256 karma sütunu bulunmaktadır. Her bir öznitelik için ilgili satırdaki metinde birbirini takip eden 2'li kelime kombinasyonları(2-gram)

karma sütunlarına sistem tarafından yerleştirilir. Bir karma sütununda ilgili kelime kombinasyonun olup olmadığı ya da kaç tane kelime kombinasyonunun olduğu, o sütunun değerinden anlaşılır.

4.2.3. Kayıp Veri Temizleme

Veri setindeki hikaye puanı sütununda bazı verilerin kayıp olduğu görülmektedir. Veri analizi yapılırken eksik veriler sorun yaratmaktadır. İyi bir çıkarım elde etmek için eksiksiz ve kesin veriler gereklidir.

Veri seti gözlendiğinde kayıp veri sayısının az olması nedeniyle veri hazırlığı aşamasında kayıp veri içeren satırlar tamamen silinmiştir.

4.3.HİPERPARAMETRE AYARLAMA

Tahmin modelleri oluşturulurken, Derin sinir ağı regresyon modeli için Tablo 4.4'de, Güçlendirilmiş Karar ağacı regresyon modeli için de Tablo 4.5'de yer alan hiperparametre değerleri, her bir veri seti için 8'er farklı rastgele kombinasyonda bir araya getirilerek grid arama yapılmıştır.

Grid arama sürecine dahil edilecek hiperparametreler literatürdeki kullanımlara göre belirlenmiştir. Bu hiperparametreler için kullanılacak değerlerin belirlenmesinde de literatürdeki kullanımlar, modelin veri yapısı ve ön grid arama sonuçları dikkate alınmıştır. Her bir model ve veri seti için, grid aramada en düşük ortalama mutlak hata değerini veren kombinasyonlar ile ilerlenmiştir.

Değerlendirme sürecinde 10 katlı Çapraz Doğrulama Yöntemi uygulanarak, ilgili veri seti 10 farklı parçaya ayrılmış ve her iterasyonda farklı bir parçası test edilerek 10 iterasyonun hata değerlerinin ortalaması, ilgili veri setinin ortalama mutlak hata değeri olarak değerlendirmeye alınmıştır.

4.3.1. Derin Sinir Ağı Regresyon Modeli

Derin sinir ağı regresyon modeli oluşturulurken grid arama sürecine dahil edilen hiperparametreler şunlardır;

- algoritmanın öğrenme süreçlerinin iterasyon sayısı,

- modelde kullanılacak kayıp/maliyet fonksiyonunun türü,
- öğrenme sürecinin başında nöron ağırlıklarını belirleyen başlangıç değeri,
- öğrenme sırasında önceki iterasyonlardan gelen nöronlara ağırlık olarak uygulanacak momentum değeri
- ve ağın hızını ve verimini artırmak amacıyla kullanılacak normalleştirme yöntemi (bkz.Tablo 4.4).

Tablo 4-4 DSA Regresyon Modeli Grid Arama Hiperparametre Değerleri

Hiperparametre	**Değerler**
İterasyon sayısı	40, 80, 160
Maliyet fonk.	Cross Entropy, Squared Error
Öğrenme Oranı	0.01
Başlangıç ağırlığı	0.1, 0.2
Momentum	0, 0.5
Normalleştirme	MinMax, Binning

Microsoft Azure ML Studio içerisinde bazı hiperparametreler özelinde kullanılabilen otomatik hiperparametre ayarlama modülü bulunmaktadır. Bu modül aracılığıyla grid arama belirlenen değerler arasında otomatik olarak yapılabilmektedir. Sinir ağı regresyon modelinde öğrenme oranı, momentum ve başlangıç ağırlığı hiperparametreleri otomatik hiperparametre ayarlama modülüne dahil edilmediği için, bu sürece manuel müdahale gerçekleştirilmiştir. Bu hiperparametrelerin de opsiyonlarını içeren alternatif modeller uygulama sürecine eklenerek, grid aramaya katılımları sağlanmıştır. Böylece bu aşamada otomatik ve manuel hiperparametre ayarlama yaklaşımları birlikte kullanılmıştır.

Microsoft Azure ML Studio'nun varsayılan sinir ağı regresyon modülünde tek gizli katman bulunmaktadır. Ancak veri setinin büyüklüğü ve karmaşıklığı nedeniyle tek katman bu uygulama için yeterli olmayacaktır.

Bu aşamada karmaşık sinir ağı mimarilerini tanımlamak için Microsoft tarafından geliştirilen bir dil olan Net # ile modül içerisine skript yazılarak, katmanlardaki nöron sayılarının öğrenme sürecinde otomatik belirleneceği, gizli katman sayısının da 2 olarak tanımlandığı bir düzenleme yapılmıştır.

4.3.2. Güçlendirilmiş Karar Ağacı Regresyon Modeli

Güçlendirilmiş karar ağacı regresyon modeli oluşturulurken, grid arama sürecine dahil edilen hiperparametreler şunlardır; Ağaç başına maksimum yaprak sayısı, Yaprak düğümü başına minimum örnek sayısı, Öğrenme oranı ve Oluşturulan ağaç sayısı.

Güçlendirilmiş karar ağacı(GKA) hiperparametrelerinin değer aralıklarının model üzerindeki riskli ve avantajlı etkilerini gözeterek, en düşük hata oranını veren hiperparametre değerlerinden oluşan kombinasyonları belirleyebilmek için sık kullanılan hiperparametreler üzerine grid arama yapılmıştır.

Tablo 4-5 GKA Regresyon Modeli Grid Arama Hiperparametre Değerleri

Hiperparametreler	**Değerler**
Ağaç başına Yaprak say.	2, 8, 32
Yaprak başına Örnek say.	10, 50
Öğrenme Oranı	0.01
Ağaç sayısı	20, 50, 100

Microsoft Azure ML Studio otomatik hiperparametre ayarlama modülü kullanılarak bu hiperparametrelerin tamamı Tablo 4.5'deki değerlere göre modellenmiştir ve her bir veri seti için en düşük hata değerlerini veren kombinasyonlar belirlenmiştir.

4.4. UYGULAMA VE ANALİZ SONUÇLARI

Bu aşamada her bir veri seti için, Tablo 4.4 ve Tablo 4.5'deki hiperparametre değerlerine göre, model bazında 8'er ayrı kombinasyonda grid arama yapılarak, en düşük hata oranını veren hiperparametre kombinasyonu listelenecektir. Daha sonra bu kombinasyonlarla tahmin modellemesi yapılarak her bir veri seti için 5 farklı performans metriğine göre sonuçlar karşılaştırılacak ve sonuçlar yorumlanacaktır.

Uygulamada kullanılan 16 veri seti şöyledir;
Aptana Studio - APT, Appcelerator Studio - AS, Bamboo - BAM, Clover - CLOV, Data Management - DM, DuraCloud - DC, JIRA Software - JSW, Apache Mesos - MES, Moodle -

MDL, Mule - MULE, Mule Studio - MS, Spring XD - SXD, Talend Data Quality - TDQ, Talend ESB - TESB, Titanum SDK/CLI - TITAN, Apache Usergrid - UG.

4.4.1. Grid Arama Sonuçları

Tablo 4.6.'de DSA(Derin Sinir Ağları) Regresyon Modeli için gerçekleştirilen grid arama sonucu veri seti bazında en düşük hata değerini veren hiperparametre kombinasyonları yer almaktadır. Tablo incelendiğinde, bazı hiperparametrelerin veri setine göre karışık değerler aldığı, bazılarının ise tüm veri setlerinde aynı değeri aldığı görülmektedir.

Tablo 4-6 DSA Regresyon Modeli, Veri Seti Bazında Grid Arama Sonuçları

H.parametreler	**APT**	**AS**	**BAM**	**CLOV**
İterasyon sayısı	80	160	160	80
Maliyet fonk.	CrossE	SquardE	SquardE	CrossE
Öğrenme Oranı	0.01	0.01	0.01	0.01
Başlangıç ağırlığı	0.1	0.1	0.1	0.1
Momentum	0.5	0.5	0.5	0.5
Normalleştirme	MinMax	MinMax	MinMax	MinMax
H.parametreler	**DM**	**DC**	**JSW**	**MES**
İterasyon sayısı	160	160	80	160
Maliyet fonk.	SquardE	SquardE	SquardE	SquardE
Öğrenme Oranı	0.01	0.01	0.01	0.01
Başlangıç ağırlığı	0.1	0.1	0.1	0.1
Momentum	0.5	0.5	0.5	0.5
Normalleştirme	MinMax	MinMax	MinMax	MinMax
H.parametreler	**MDL**	**MULE**	**MS**	**SXD**
İterasyon sayısı	80	80	80	40
Maliyet fonk.	CrossE	CrossE	CrossE	CrossE
Öğrenme Oranı	0.01	0.01	0.01	0.01
Başlangıç ağırlığı	0.1	0.1	0.1	0.1
Momentum	0.5	0.5	0.5	0.5
Normalleştirme	MinMax	MinMax	MinMax	MinMax
H.parametreler	**TDQ**	**TESB**	**TITAN**	**UG**
İterasyon sayısı	160	160	80	160
Maliyet fonk.	SquardE	SquardE	SquardE	SquardE
Öğrenme Oranı	0.01	0.01	0.01	0.01
Başlangıç ağırlığı	0.1	0.1	0.1	0.1
Momentum	0.5	0.5	0.5	0.5
Normalleştirme	MinMax	MinMax	MinMax	MinMax

Tüm veri setlerinde başlangıç ağırlığının 0.1 olduğu, momentumun kullanıldığı ve normalleştirme türünün Min-Max olduğu hiperparametre değerleri en iyi sonucu vermiştir.

Böylece bu hiperparametreler için grid aramadaki diğer seçeneklerin model üzerinde iyi sonuçlar vermediği söylenebilir.

40 iterasyonun hiçbir veri seti için yeterli olmadığı açık olmakla beraber, 80 ve 160 iterasyonun, veri setlerinde yarı yarıya en iyi sonucu verdiği görülmüştür. Kare Hata amaç fonsiyonunun, veri setlerinin çoğunda en iyi değerleri verdiği gözlemlenmiştir. Ancak bazı veri setlerinde Çapraz Entropi fonksiyonu da en düşük hata değerini vermiştir.

Tablo 4.7'da GKA Regresyon Modeli için gerçekleştirilen grid arama sonucu veri seti bazında en düşük hata değerini veren hiperparametre kombinasyonları yer almaktadır. Tablo incelendiğinde, bir yaprak oluşturmak için gereken vaka sayısı yüksek olduğunda, bir ağaçta oluşturulabilecek yaprak sayısının makul derecede düşük olduğu modellerin en iyi model olarak seçildiği gözlemlenmiştir.

Tablo 4-7 GKA Regresyon Modeli, Veri Seti Bazında Grid Arama Sonuçları

H.parametre	**APT**	**AS**	**BAM**	**CLOV**	**DM**	**DC**	**JSW**	**MES**
Ağaç başına Yaprak say.	32	32	8	8	32	8	32	8
Yaprak başına Örnek say.	10	10	50	50	10	50	10	50
Öğrenme Oranı	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Ağaç sayısı	100	100	100	100	100	100	100	100
H.parametre	**MDL**	**MULE**	**MS**	**SXD**	**TDQ**	**TESB**	**TITAN**	**UG**
Ağaç başına Yaprak say.	32	8	32	32	8	8	32	8
Yaprak başına Örnek say.	10	50	10	10	50	50	10	50
Öğrenme Oranı	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Ağaç sayısı	100	100	100	100	100	100	100	100

20 ve 50 ağaç oluşturmanın hiçbir veri setinde iyi sonuç vermediği görülmekle birlikte, tüm veri setlerinde 100 ağaç sayısı en düşük hata değerini vermiştir. Bir yaprak oluşturmak için gereken örnek sayısı, veri setlerinin yarısında 10, diğer yarısında 50 olarak alındığında grid arama en iyi sonuçları vermiştir. Ağaç başına yaprak sayısı, veri setlerinin yarısında 8 diğer yarısında 32 olduğunda grid aramadaki en iyi modellere ulaşılmıştır. Ağaç başına 2 yaprak, hiçbir veri seti için iyi sonuç vermemiştir.

4.4.2. Karşılaştırmalı Değerlendirme Sonuçları

Bu aşamaya kadar geçen süreçte hazırlık işlemlerinden geçen veri setlerinin her biri üzerine grid arama gerçekleştirilerek en düşük hata oranlarını veren hiperparametre değerleri belirlenmiştir. Bu hiperparametre değerleri kullanılarak her bir veri seti üzerine 5 farklı performans değerlendirme metriği üzerinden sonuçlar elde edilmiştir. Bu sonuçlar Derin Sinir Ağı Regresyon (DSAR) modeli ve Güçlendirilmiş Karar Ağacı Regresyon (GKAR) modeli için karşılaştırılarak hangi modelin hangi veri setinde daha etkin sonuçlar verdiği belirlenmiştir.

Tablo 4.8'de, bu çalışmada oluşturulan Derin Sinir Ağı Regresyon Modeli ve Güçlendirilmiş Karar Ağacı Regresyon modelinin 16 ayrı veri seti üzerine Çapraz Doğrulama Modeliyle elde edilen karşılaştırmalı sonuçları sunulmuştur.

Tablo 4-8 DSAR ve GKAR Karşılaştırmalı Değerlendirme Sonuçları, MAE, RMSE, RAE, RSE, COD

Veri Seti	Yöntem	MAE
APTANA	**DSAR**	**0.96**
	GKAR	3.12
AS	**DSAR**	**0.65**
	GKAR	2.35
BAM	**DSAR**	**0.74**
	GKAR	0.90
CLOV	**DSAR**	**1.75**
	GKAR	2.38
DM	**DSAR**	**1.58**
	GKAR	3.72
DC	**DSAR**	**0.59**
	GKAR	0.77
JSW	**DSAR**	**1.24**
	GKAR	1.95
MESOS	**DSAR**	**0.83**
	GKAR	1.37
MDL	**DSAR**	**1.49**
	GKAR	6.03
MULE	**DSAR**	**1.03**
	GKAR	2.16
MS	**DSAR**	**1.37**
	GKAR	2.70
SXD	**DSAR**	**0.86**
	GKAR	1.56
TDQ	**DSAR**	**1.25**
	GKAR	2.53
TESB	**DSAR**	**0.44**
	GKAR	0.79
TITAN	**DSAR**	**0.88**
	GKAR	2.60
UG	**DSAR**	**0.59**
	GKAR	1.06

Veri Seti	Yöntem	RMSE
APTANA	**DSAR**	**1.73**
	GKAR	4.60
AS	**DSAR**	**1.45**
	GKAR	3.14
BAM	**DSAR**	**1.38**
	GKAR	1.77
CLOV	**DSAR**	**3.05**
	GKAR	5.24
DM	**DSAR**	**5.58**
	GKAR	8.82
DC	**DSAR**	**1.12**
	GKAR	1.53
JSW	**DSAR**	**2.13**
	GKAR	2.99
MESOS	**DSAR**	**1.69**
	GKAR	2.28
MDL	**DSAR**	**3.06**
	GKAR	10.05
MULE	**DSAR**	**1.57**
	GKAR	2.67
MS	**DSAR**	**2.15**
	GKAR	3.52
SXD	**DSAR**	**1.65**
	GKAR	2.43
TDQ	**DSAR**	**2.01**
	GKAR	3.49
TESB	**DSAR**	**0.88**
	GKAR	1.24
TITAN	**DSAR**	**1.64**
	GKAR	3.35
UG	**DSAR**	**1.02**
	GKAR	1.40

Veri Seti	Yöntem	RAE
APTANA	**DSAR**	**0.23**
	GKAR	0.76
AS	**DSAR**	**0.24**
	GKAR	0.87
BAM	**DSAR**	**0.61**
	GKAR	0.73
CLOV	**DSAR**	**0.49**
	GKAR	0.61
DM	**DSAR**	**0.16**
	GKAR	0.39
DC	**DSAR**	**0.48**
	GKAR	0.62
JSW	**DSAR**	**0.51**
	GKAR	0.79
MESOS	**DSAR**	**0.40**
	GKAR	0.66
MDL	**DSAR**	**0.11**
	GKAR	0.44
MULE	**DSAR**	**0.38**
	GKAR	0.81
MS	**DSAR**	**0.38**
	GKAR	0.75
SXD	**DSAR**	**0.40**
	GKAR	0.72
TDQ	**DSAR**	**0.32**
	GKAR	0.66
TESB	**DSAR**	**0.42**
	GKAR	0.75
TITAN	**DSAR**	**0.24**
	GKAR	0.72
UG	**DSAR**	**0.62**
	GKAR	1.12

Veri Seti	Yöntem	RSE
APTANA	**DSAR**	**0.23**
	GKAR	0.61
AS	**DSAR**	**0.15**
	GKAR	0.71
BAM	**DSAR**	**0.60**
	GKAR	0.76
CLOV	**DSAR**	**0.32**
	GKAR	0.74
DM	**DSAR**	**0.11**
	GKAR	0.28
DC	**DSAR**	**0.35**
	GKAR	0.61
JSW	**DSAR**	**0.44**
	GKAR	0.80
MESOS	**DSAR**	**0.36**
	GKAR	0.65
MDL	**DSAR**	**0.02**
	GKAR	0.22
MULE	**DSAR**	**0.21**
	GKAR	0.60
MS	**DSAR**	**0.16**
	GKAR	0.45
SXD	**DSAR**	**0.27**
	GKAR	0.58
TDQ	**DSAR**	**0.15**
	GKAR	0.45
TESB	**DSAR**	**0.35**
	GKAR	0.70
TITAN	**DSAR**	**0.10**
	GKAR	0.44
UG	**DSAR**	**0.56**
	GKAR	1.04

Veri Seti	Yöntem	COD
APTANA	**DSAR**	**0.08**
	GKAR	0.38
AS	**DSAR**	**0.84**
	GKAR	0.28
BAM	**DSAR**	**0.39**
	GKAR	0.23
CLOV	**DSAR**	**0.67**
	GKAR	0.25
DM	**DSAR**	**0.88**
	GKAR	0.71
DC	**DSAR**	**0.64**
	GKAR	0.38
JSW	**DSAR**	**0.55**
	GKAR	0.19
MESOS	**DSAR**	**0.63**
	GKAR	0.34
MDL	**DSAR**	**0.97**
	GKAR	0.77
MULE	**DSAR**	**0.78**
	GKAR	0.39
MS	**DSAR**	**0.83**
	GKAR	0.54
SXD	**DSAR**	**0.72**
	GKAR	0.41
TDQ	**DSAR**	**0.84**
	GKAR	0.54
TESB	**DSAR**	**0.64**
	GKAR	0.29
TITAN	**DSAR**	**0.89**
	GKAR	0.55
UG	**DSAR**	**0.43**
	GKAR	-0.04

Uygulama çalıştırıldıktan sonra tüm sonuçların elde edilmesi toplam 651 dakika sürmüştür. Performans değerlendirme metriği olarak MAE (Ortalama Mutlak Hata) değerleri karşılaştırıldığında, DSAR modelleri arasında CLOV veri seti en yüksek hatayı verirken, TESB veri seti en düşük hatayı vermiştir. GKAR modelleri arasında ise en düşük hatayı veren veri seti DC iken en yüksek hatayı veren veri seti MDL olmaktadır. Bu anlamda iki modelin en iyi ve en kötü sonuçlar verdiği veri setlerinin farklı olduğu görülmektedir.

MDL veri setinin GKAR modelindeki MAE oranı tüm sonuçlar arasında bir hayli yüksektir. MDL aynı zamanda DSA modelinde determinasyon katsayısının da en yüksek olduğu veri setidir. Yine tüm sonuçlar ele alındığında en düşük hata değerini veren veri seti ise DSAR modeliyle TESB veri seti olmaktadır.

İki modelin sonuçları karşılaştırıldığında GKAR modelinin MAE değerlerinin ortalaması 2.25 iken DSAR modelinin MAE değerleri ortalaması 1.02 olarak hesaplanmıştır. Ayrıca tüm veri setleri için DSAR modelinin daha düşük MAE oranları verdiği gözlemlenmiştir. Sonuç olarak DSAR modelinin çok daha güçlü sonuçlar verdiği görülmektedir. Tezde önerilen Derin Ağı Regresyon yönteminin, veri setleri bazında MAE oranları 0,44 ile 1,75 arasında değişmektedir. Veri setlerindeki hikaye puanlarının yaklaşık 1 ila 40 arasında değişmekte olduğu düşünüldüğünde, 1,02 MAE değerleri ortalamasıyla DSAR modelinin oldukça etkili bir model olduğu görülmektedir.

5. TARTIŞMA VE SONUÇ

Bu çalışmada derin öğrenme tabanlı bir tahmin yönteminin, çevik proje yönetiminde efor tahmini problemlerine uygulanışı incelenmiştir. Çalışmada yazılım proje yönetiminde kritik öneme sahip bir adım olan, ancak çoğunlukla manuel ve uzman yargısına dayalı olarak yapılan efor tahmininin daha hızlı, daha tutarlı ve daha tekrar edilebilir ölçüde yapılabilmesi için derin öğrenme tabanlı bir yaklaşım uygulanmıştır. Bu doğrultuda çevik proje yönetimi uygulayan yazılım firmalarına hem zamandan ve maliyetten tasarruf edebilecekleri, hem de tutarlı ve sağlıklı tahminler elde edebilecekleri derin öğrenme tabanlı otomatik bir sistem kullanma önerisi sunulmuştur.

Bu yaklaşımın etkisinin artırılabilmesi için öğrenme sürecinden önce veri bir dizi hazırlık sürecinden geçirilmiştir. Uygulama için son yıllarda yazılım projelerinin büyük çoğunluğunda uygulanan çevik proje yönetim metodolojisine dayanan 16 adet açık kaynaklı veri seti seçilmiştir.

Bu çalışma, kullanıcılardan gelen metin bazlı veriyi derin öğrenme tabanlı bir yaklaşımla çevik proje yönetiminde efor tahmini yapabilmek için, öznitelik çıkarımı dahil bir dizi veri hazırlık sürecinden geçirerek uygulamasıyla ve güçlendirilmiş karar ağacı regresyon yöntemiyle karşılaştırmasıyla literatüre yenilikçi bir çalışma kazandırmıştır.

Tezin ilk bölümünde yazılım proje yönetiminden, yazılım projelerinde efor tahminin öneminden ve literatürdeki örneklerinden bahsedilmiştir. Ayrıca tezin uygulama sürecindeki temel süreçler olan Veri ve Metin madenciliği, Derin Sinir Ağları ve Karar Ağaçları genel çerçevesiyle anlatılmıştır. İkinci bölümde veri hazırlığı ve hiperparametre ayarlama süreçleri açıklanarak, DSA (Derin Sinir Ağları) ve GKA (Güçlendirilmiş Karar Ağacı) yaklaşımlarında kullanılan temel hiperparametreler detaylandırılmıştır. Üçüncü bölümde uygulama sürecinde hangi işlemlerin gerçekleştirildiği adım adım anlatılmıştır. Bu kapsamda veri hazırlığı aşamasında yapılan işlemler ve hiperparametre ayarlama sürecinde kullanılacak grid arama değerleri gösterilmiştir. Efor tahmini metriği olarak çevik proje yönetimi uygulayan yazılım projelerinde kullanılan hikaye puanı baz alınmıştır.

Dördüncü bölümde bu yaklaşımla elde edilen sonuçlar bir başka makine öğrenmesi tekniği olan güçlendirilmiş karar ağacı yönteminin sonuçlarıyla karşılaştırılmıştır. İki modelin sonuçları karşılaştırıldığında DSAR (Derin Sinir Ağları Regresyon) modelinin MAE (Ortalama Mutlak Hata) değerleri ortalamasının, GKAR (Güçlendirilmiş Karar Ağacı Regresyon) modelinin MAE değerleri ortalamasının yarısından az olduğu görülmüştür. Ayrıca tüm veri setleri için DSAR modelinin daha düşük MAE oranları verdiği gözlemlenmiştir. Sonuç olarak DSAR modelinin çok daha güçlü sonuçlar verdiği görülmektedir.

Proje eforunun doğru tahmin edilmesi proje yönetiminin başarısını doğrudan etkilemektedir. Yanlış efor tahminiyle projeye başlamak, proje sonunda firma için prestij ve para kaybına yol açabilmektedir. Bu nedenle uygulanan tahmin yöntemi oldukça önemlidir. Yapılan bazı istatistikler, yazılım projelerinde son yıllarda kullanılan en yaygın proje yönetim metodolojisinin çevik yöntem olduğunu göstermektedir. Çevik proje yönetim metodolojisini uygulayan firmaların büyük çoğunluğunda tahminler insan yargısına dayalı ve manuel yapılmakta ve yapılan tahminler birim yöneticilerinin kontrolünden geçmektedir. Çevik projelerdeki tahmin kalemlerinin fazlalığı dikkate alındığında, bu yöntemde fazlasıyla efor harcandığı açıkça görülmektedir. Bunun yanı sıra tahminin sezgisel bir yol izlenerek yapılması nedeniyle tahmin tutarlılığı düşük olabilmektedir.

Bu çalışmada derin öğrenme tabanlı bir yaklaşım kullanılarak otomatik bir sistem önerilmiştir. Bu sayede tahmin süreçleri çok daha az insan eforu harcayarak hızlı, tarafsız ve tutarlı şekilde gerçekleştirilebilecektir.

Bu yaklaşım farklı veri setlerine de uygulanabilir bir yaklaşım olmakla birlikte, veri hazırlığı ve model parametrelerini belirleme aşamaları ilgili veri setinin yapısına göre değişiklik göstereceğinden uğraş gerektirecek süreçler içerebilecektir. Bu nedenle firmada veri seti ve proje bazında bu süreçleri takip edecek ve geliştirecek uzman bir çalışanın bulunması ya da bu desteğin dışarıdan alınması gerekmektedir. Bunun yanı sıra model karmaşıklığı arttıkça CPU ihtiyacı da artacağından ilgili donanım ihtiyaçlarının karşılanması da gerekmektedir.

Bu çalışmanın bir sonraki adımında, farklı bir proje yönetim metodolojisi uygulayan, dolayısıyla farklı ölçü birimlerinin ve veri çeşitlerinin kullanıldığı bir projede efor tahmin çalışması yapılabilir. Bunun dışında karşılaştırma için bu tezde kullanılan yöntemlere, başka

makine öğrenmesi yöntemleri de eklenebilir ve veri hazırlığı aşamasında yine ilgili veri setine göre daha farklı işlemler uygulanabilir.

KAYNAKLAR

Agrawal V., Shrivastava V., 2015, Performance Evaluation of Software Development Effort Estimation Using Neuro-Fuzzy Model, *International Journal of Emerging Research in Management &Technology 2015 (47) pp: 2278-9359*

Aguirre D., Fuentes O., 2019, Improving Weight Initialization of ReLU and Output Layers, *Artificial Neural Networks and Machine Learning – ICANN 2019: Deep Learning pp 170-184*

Almquist I., Lindblom E., 2019, Data-Driven Predictions of Heating Energy Savings in Residential Buildings, *Sweden Uppsala University, ISSN: 1650-8319, UPTEC STS 19027*

Ananthakumar K., Shivakumar N., Balaji N., 2016, A Neuro Fuzzy Algorithm to Compute Software Effort Estimation, *Global Journal of Computer Science and Technology: C Software & Data Engineering vol: 16*

Anderson, D. ve Mc Neill, G., 1992, Artificial neural networks, *Technology, Kaman Sciences Corporation, New York*

Arbain S.H., Ali N.A., Mustaffa N. H., 2019, Adoption of Machine Learning Techniques in Software Effort Estimation: An Overview, *IOP Conference Series: Materials Science and Engineering , Joint Conference on Green Engineering Technology & Applied Computing 2019*

Barati Z., Shahbarzade M., Bardsiri V., 2016, Type-2 Fuzzy Logic Approach To Increase The Accuracy Of Software Development Effort Estimation, *Journal of Advances in Computer Engineering and Technology, 2(4) 2016*

Bedi R.P.S., Singh A., 2010, Software cost estimation using fuzzy logic, *Indian Journal of Science and Technology, Vol 10(3), DOI: 10.17485/ijst/2017/v10i3/109997, January 2017*

Bengio Y., 2000, Continuous optimization of hyper-parameters, *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*

Berry M.J.A, Linoff G., 1997, *Data Minning Techniques For Marketing, Sales, and Customer Support,* ISBN:0471179809 John Wiley & Sons, Inc. New York, NY, USA

Braga P., Oliveira A., Meira S., Software Effort Estimation using Machine Learning Techniques with Robust Confidence Intervals , *ICTAI 2007. 19th IEEE International Conference on, vol.1, IEEE, 2007*

Buduma N., Locascio N., 2017, Fundamentals of Deep Learning: Design- ing Next-Generation Machine Intelligence Algorithms. Sebastopol, *CA: O'Reilly Media, first ed.*

Caragea C., Silvescu A., Mitra P., 2012, Combining Hashing and Abstraction in Sparse High Dimensional Feature Spaces, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*

Choetkiertikul M., Dam H.K, Tran T., Pham T., Ghose A., Menzies T., 2019, A deep learning model for estimating story points. *IEEE Transactions on Software Engineering, 45 (7), 637-656.*

Collados J. C., Pilehvar M.T., 2017, On the Role of Text Preprocessing in Neural Network Architectures: An Evaluation Study on Text Categorization and Sentiment Analysis, *arXiv preprint, (2017): 1707.01780*

Dafni R.J., 2014, An ontology based approach for text clustering using association rule mining and must link and cannot link algorithm, *Phd Thesis, Faculty of Information and Communication Engineering, Anna University*

Dejaeger K., Verbeke W., Martens D., Baesens B., 2012, Data mining techniques for software effort estimation: A comparative study, *Ieee Transactions on Software Engineering, Vol. 38, No. 2, March/April 2012*

Dixit, A., 2017, Ensemble Machine Learning, *Birmingham, UK, Packt Publishing Ltd.*

Djouadi Y., Souam F., 2007, Bottom Up Text Mining through Hierarchical Document Representation, *World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering Vol:1, No:11*

Ebrahimpour N., Gharehchopogh F.S, , Khalifehlou Z.A., 2016, New Approach with Hybrid of Artificial Neural Network and Ant Colony Optimization in Software Cost Estimation, *Journal of Advances in Computer Research Quarterly eISSN: 2345-6078 Vol. 7, No. 4, November 2016, Pages: 1-12*

Efe M.Ö., Kaynak O., 2000, Yapay Sinir Ağları ve Uygulamaları, *Boğaziçi Üniversitesi Yayınevi,* İstanbul

Ferdiana R., Santoso R.I., Nugroho L.E., Ashari A., 2011, User Story Software Estimation: a Simplification of Software Estimation Model with Distributed Extreme Programming Estimation Technique, *Jurnal Ilmiah Teknologi Informasi, Volume 9, Nomor 1, Januari 2011: 41 – 48*

Ferreira P.P., 2016, Volume and Price in The Nordic Balancing Power Market, *Ms. Thesis, Norwegian University of Science and Technology, Department of Electric Power Engineering*

Gharipour A., Sameti M., Yousefian A., 2010, A Comparative Approximate Economic Behavior Analysis of Support Vector Machines and Neural Networks Models, *Iranian Economic Review, Vol.15, No.26*

Gritta R. D., Davalos S., 2002, Forecasting Small Air Carrier Bankruptcies Using A Neural Network Approach: A Preliminary Analysis, *Journal of Financial Management and Analysis, Vol. 13, No.1, 44-49*

Hernandez L.J.M., Adams R.P., 2015, Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks, *School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138 USA*

Hovden I. T., 2019, Optimizing Artificial Neural Network Hyperparameters and Architecture, *Faculty of Mathematics and Natural Sciences, Department of Physics, University of Oslo, IN9400*

https://www.mathworks.com/discovery/machine-learning.html) [Ziyaret Tarihi: 06.04.2020]

Humayun M., Gang C., 2012, Estimating Effort in Global Software Development Projects Using Machine Learning Techniques, International Journal of Information and Education Technology*, Vol. 2, No. 3, June 2012*

Işıkhan S.Y., 2014, Mikrodizilim Gen İfade Çalışmalarında Genelleştirme Yöntemlerinin Regresyon Modelleri Üzerine Etkisi, *Ph.D. Thesis, Hacettepe University Institute of Health Science, in Biostatistics*

Jain S., Shukla S., Wadhvani R., 2018, Dynamic selection of Normalization Techniques using Data Complexity Measures, *Exp. Syst. Appl. 106, 252–262*

Kangwantrakool T., Viriyayudhakorn K., Theeramunkong T., 2020, Software Development Effort Estimation from Unstructured Software Project Description by Sequence Models, *IEICE TRANS. INF. & SYST., VOL.E103–D, NO.4 APRIL 2020*

Karsoliya S., Approximating Number of Hidden layer neurons in Multiple Hidden Layer BPNN Architecture*, International Journal of Engineering Trends and Technology- Volume 3 Issue 6- 2012*

Kavakiotis I., Tsave O., Salifoglou A., Maglaveras N., Vlahavas I., Chouvarda I, 2017, Machine Learning and Data Mining Methods in Diabetes Research, *Computational and Structural Biotechnology Journal 15 (2017) 104– 116*

Khuat T., Le T.M.H., 2017, Applying teaching-learning to artificial bee colony for parameter optimization of software effort estimation model, *Journal of Engineering Science and Technology12(5):1178 - 1190*

Kim J., Kim J., Jang G., Lee M., 2017, Fast learning method for convolutional neural networks using extreme learning machine and its application to lane detection, *School of Electronics Engineering, Kyungpook National University Sankyuk-Dong, Puk-Gu, Taegu 702-701, Korea*

Kosinar M., Strba R., 2013, Simulations of Agile Software Processes for Healthcare Information Systems Development Based on Machine Learning Methods, *12th IFAC Conference on Programmable Devices and Embedded Systems, The International Federation of Automatic Control, September 25-27. Velke Karlovice, Czech Republic*

Kumar M, Raghuwanshi N. S.,Singh R., Wallender W. W., Pruitt W. O., 2002, Estimating Evapotranspiration using Artificial Neural Network, *10.1061/(ASCE)0733-9437(2002)128:4(224)*

Laqrichi S., Marmier F., Gourc D., Nevoux J.,Integrating uncertainty in software effort estimation using Bootstrap based Neural Networks, *IFAC*-PapersOnline 48-3 (2015) pp: 954-959

LeCun, Y., Bengio, Y., Hinton, G., 2015, Deep Learning., *Nature, 521, 436-444.*

Li Z., Yu X., 2015, Software project management based on dynamic programming algorithm, *International Conference on Information Sciences, Machinery, Materials and Energy (ICISMME 2015)*

Lin W., Danny D., Luiz H., Capretz F., 2010, Improving Software Effort Estimation Using Neuro-Fuzzy Model with SEER-SEM, *Global Journal of Computer Science and Technology Vol. 10 Issue 12 (Ver. 1.0) October 2010*

Lucas, C., Nielsen, R., Roberts, M., Stewart, B., Storer, A., Tingley, D., 2015, Computer assisted text analysis for comparative politics. *Political Analysis 23 (2) 254-277*

MacDonell S., Shepperd M., 2003, Combining techniques to optimize effort predictions in software project management, *The Journal of Systems and Software 66 (2003) 91–98*

Mahmood Y., Kama N., Azmi A., 2019, A systematic review of studies on use case points and expert-based estimation of software development effort, *Journal of Software Evolution and Process* 2020

Malhotra R., Jain A., 2011, Software Effort Prediction using Statistical and Machine Learning Methods, *(IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 2, No.1*

Mehdiyev N., Fettke P., Evermann J., 2017, A Multi-stage Deep Learning Approach for Business Process Event Prediction, *2017 IEEE 19th Conference on Business Informatics*

Mendenhall J., and Meiler J., 2016, Improving Quantitative Structure-Activity Relationship Models using Artificial Neural Networks Trained with Dropout, *Journal of Computer-Aided Molecular Design 30, 177–189*

Michael A. Nielsen, , 2015, Neural Networks and Deep Learning, *Determination Press, 2015*

Microsoft Azure Documentation, https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/ [Ziyaret tarihi: 01.04.2020]

Milicevic M., 2019, Integration of big data analysis and engineering advanced analytics to intelligently manage and control completion and artifical lift for selected well in Croatia, *Master's thesis, University of Zagreb, Faculty of Mining, Geology and Petroleum Engineering, Department of Petroleum Engineering*

Nassif A.B., Talib M.A., Capretz L.F., 2020, Software Effort Estimation from Use Case Diagrams Using Nonlinear Regression Analysis, *arXiv preprint, (2020): 2003.09873*

Nwaiwu J., 2016, 22 Analytic Study of Fuzzy-based model for Software Cost Estimation, *CoRI'16, Sept 7–9, 2016, Ibadan, Nigeria*

Nwaiwu J., 2016, 22 Analytic Study of Fuzzy-based model for Software Cost Estimation, *CoRI'16, Sept 7–9, 2016, Ibadan, Nigeria*

Olgun S., 2009, Tedarik Zinciri Yönetiminde Talep Tahmini Yöntemleri ve Yapay Zeka Tabanlı Bir Talep Tahmin Modelinin Uygulanması, *YL Tezi, İstanbul Üniversitesi, Endüstri Mühendisliği Bölümü*

Owais M, Ramakishore R, 2016, Effort, Duration and Cost Estimation in Agile Software Development*, 2016 Ninth International Conference on Contemporary Computing (IC3)*

Panchal F.S., Panchal M., 2014, Review on Methods of Selecting Number of Hidden Nodes in Artificial Neural Network, *IJCSMC, Vol. 3, Issue. 11, November 2014, pg.455 – 464*

Panchal G., Ganatra A., Kosta Y.P., Panchal D., 2011, Behaviour Analysis of Multilayer Perceptrons with Multiple Hidden Neurons and Hidden Layers, *International Journal of Computer Theory and Engineering, Vol. 3, No. 2, April 2011 ISSN: 1793-8201*

Pandya A. ve Macy R., 1996, Pattern recognition with neural networks in C++, *CRCPres, ABD, 0849394627*

Patterson J., Gibson A., 2017, Deep Learning: A Practitioner's Approach, *65-71*

Perovsek M., Kranjc J., Erjavec T., Cestnik B., Lavracˇ N., 2016, TextFlows: A visual programming platform for text mining and natural language processing, *Science of Computer Programming 121 (2016) 128–152*

PERSSON S., 2018, Application of the german traffic sign recognition benchmark on the vgg16 network using transfer learning and bottleneck features in keras, *Master Thesis in Engineering Physics, Sweden Uppsala University*

Porru S., Murgia A., Demeyer S., Marchesi M., Tonelli R., 2016, Estimating Story Points from Issue Reports, *PROMISE 2016: Proceedings of The 12th International Conference on Predictive Models and Data Analytics in Software Engineering, September 2016, Article No.: 2 Pages 1–10*

Rijwani P., Jain S., 2016, Enhanced Software Effort Estimation using Multi Layered Feed Forward Artificial Neural Network Technique, *Procedia Computer Science, Volume 89, 2016, Pages 307-312*

Rumelhart, D. E., Hinton, G. E., Williams, R. J., 1986, Learning representations by back-propagating errors, *Nature, 323, 533-536.*

Savic M., 2009, Multi-parameter based diagnosis of lung disease, *United States Patent Application Publication, US2009/0156950A1*

Sayad S., https://www.saedsayad.com/model_evaluation_r.htm [Ziyaret Tarihi:06.04.2020]

Scott E., Pfahl D., 2018, Using Developers' Features to Estimate Story Points, *ICSSP '18: Proceedings of the 2018 International Conference on Software and System Process*

Shahul S., Suneel S., Rahaman M.A, Swathi J.N, 2016, Study of Data Pre-Processing Techniques for Machine Learning Algorithm to Predict Software Effort Estimation*, Imperial Journal of Interdisciplinary Research (IJIR) Vol-2, Issue-6, 2016 ISSN: 2454-1362*

Sheila K. G.,Deepak S. N., Review on Methods to Fix Number of Hidden Neurons in Neural networks, *Mathematical Problems in Engineering, vol. 2013, Article ID 425740, 11 pages, 2013*

Smith L.N., 2018, A disciplined approach to neural network hyper-parameters: Part 1 -- learning rate, batch size, momentum, and weight decay, *US Naval Research Laboratory Technical Report 5510-026*

Soner T.,2014, Parametrik Tahmin Modellerinin Yazılım Projelerine Uygulanmasına Yönelik Bir Yazılım Paketinin Geliştirilmesi, *YL Tezi, Ankara Üniversitesi, Fen Bilimleri Enstütüsü*

Stamey, T., Kabalin, J.N., McNeal, J.E., Johnstone, I.M., Freiha, F., Redwine, E.A., 1989, Prostate specific antigen in the diagnosis and treatment of adenocarcinoma of the prostate. *II. Radical prostatectomy treated patients, Journal of Urology, 16, 1076-1083.*

Stoica M., Mircea M., Ghilic-Micu B., 2013, Software Development: Agile vs. Traditional, *Informatica Economică vol. 17, no. 4/2013*

Taylor J.G., 2005, The promise of neural networks, *Springer-Verlag , 1-131*

Versionone, 2019, 13th Annual State of Agile Survey, *Retrieved from https://www.stateofagile.com/#ufh-i-521251909-13th-annual-state-of-agile- report/473508*

Williams A., 2014, Estimating User Interaction Probability for Non-Guaranteed Display Advertising, *Master Thesis , Department of Mathematics and Statistics, University of Canterbury*

Wong E. W. C., Kim D. K., 2018, A simplified method to predict fatigue damage of TTR subjected to short- term VIV using artificial neural network, *Advances in Engineering Software Volume 126, December 2018, Pages 100-109*

www.towardsdatascience.com [Ziyaret tarihi: 03.03.2020]

Yangin G., 2019, Xgboost ve Karar Ağacı Tabanlı Algoritmaların Diyabet Veri Setleri Üzerine Uygulanması, *YL Tezi, Mimar Sinan Üniversitesi, Fen Bilimleri Enstütüsü, İstatistik Ana Bilim Dalı, İstanbul*

Zhang J., Ting-Jie L., Zhao Y., 2006, Study on Top-Down Estimation Method of Software Project Planning, *THE JOURNAL OF CHINA UNIVERSITIES OF POSTS AND TELECOMMUNICATIONS Vol. 13, No. 2, Jun. 2006*

Zhang Q1 L., Tan J., Han D., Zhu H., From machine learning to deep learning: progress in machine intelligence for rational drug discovery, *Drug Discovery Today 2017*

ÖZGEÇMİŞ

Kişisel Bilgiler	
Adı Soyadı	Senem Olgun
Doğum Yeri	İzmir
Doğum Tarihi	
Uyruğu	[X]T.C. □ Diğer:
Telefon	
E-Posta Adresi	2602090042@ogr.iu.edu.tr
Web Adresi	



Yüksek Lisans	
Üniversite	İstanbul Üniversitesi-Cerrahpaşa
Enstitü Adı	Lisansüstü Eğitim Enstitüsü
Anabilim Dalı	Endüstri Mühendisliği Anabilim Dalı
Programı	Endüstri Mühendisliği Programı

Doktora	
Üniversite	İstanbul Üniversitesi-Cerrahpaşa
Enstitü Adı	Lisansüstü Eğitim Enstitüsü
Anabilim Dalı	Endüstri Mühendisliği Anabilim Dalı
Programı	Endüstri Mühendisliği Programı

Makale ve Bildiriler