T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

TEKRAR EDEN VERİ ANALİZİNİ KULLANARAK YAZILIM GELİŞTİRME İÇİN İYİLEŞTİRİLMİŞ HATA TAHMİNİ

DOKTORA TEZİ

Muhammed Maruf ÖZTÜRK

Enstitü Anabilim Dalı	**:**	**BİLGİSAYAR VE BİLİŞİM MÜHENDİSLİĞİ**
Tez Danışmanı	**:**	**Doç. Dr. Ahmet ZENGİN**

Nisan 2016

T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

TEKRAR EDEN VERİ ANALİZİNİ KULLANARAK YAZILIM GELİŞTİRME İÇİN İYİLEŞTİRİLMİŞ HATA TAHMİNİ

DOKTORA TEZİ

Muhammed Maruf ÖZTÜRK

Enstitü Anabilim Dalı : BİLGİSAYAR VE BİLİŞİM MÜHENDİSLİĞİ

Bu tez 22 / 04 / 2016 tarihinde aşağıdaki jüri tarafından oybirligi / oy çokluğu ile kabul edilmiştir.

Prof. Dr. **Tuncay YİĞİT** **Jüri Başkanı**	**Prof. Dr.** **İbrahim ÇİL** **Üye**	**Doç. Dr.** **Ahmet ÖZMEN** **Üye**
Doç. Dr. **Ahmet ZENGİN** **Üye**		**Yrd. Doç. Dr.** **Arif KOYUN** **Üye**

BEYAN

Tez içindeki tüm verilerin akademik kurallar çerçevesinde tarafımdan elde edildiğini, görsel ve yazılı tüm bilgi ve sonuçların akademik ve etik kurallara uygun şekilde sunulduğunu, kullanılan verilerde herhangi bir tahrifat yapılmadığını, başkalarının eserlerinden yararlanılması durumunda bilimsel normlara uygun olarak atıfta bulunulduğunu, tezde yer alan verilerin bu üniversite veya başka bir üniversitede herhangi bir tez çalışmasında kullanılmadığını beyan ederim.

Muhammed Maruf ÖZTÜRK
03.05.2016

ÖNSÖZ

Yazılım geliştirme maliyetlerinin %50'ye yakın bölümünü yazılım bakım maliyetleri oluşturmaktadır. Yazılım sistemlerinin gün geçtikçe büyümesi ve karmaşıklaşması yeni metot ve tekniklerin ihtiyacını beraberinde getirmektedir. Bu yöntemlerden biri de yazılımlardaki tarihsel veriler incelenerek yapılan hata tahmin işlemidir. Bu tez çalışmasında yazılım hata tahmin performansını arttıran bir ön-işleme algoritması geliştirilmiş, algoritmayı ve hata veri seti performans işlemlerini de içeren bir yazılım çerçevesi sunulmuştur.

Doktora eğitimim süresince bana her türlü yardımı esirgemeyen danışmanım Doç. Dr. Ahmet ZENGİN'e ve tezin gerekli bilimsel yeterliliğe ulaşması için tecrübe ve bilgilerini aktaran tez izleme komite üyeleri hocalarım Prof. Dr. İbrahim Çil ve Doç. Dr. Ahmet Özmen'e, aileme;

Sonsuz minnet ve teşekkürlerimi sunarım.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR LİSTESİ

AUC	: Area under the curve / Eğrinin altında kalan alan
cCount	: Character Count / Karakter sayısı
cS	: Class Size / Sınıf büyüklüğü
Dnp	: None defect prone / Hata yatkın değil
Dp	: Defect prone / Hata yatkın
F-measure	: F-ölçüt değeri
FN	: False nagative
FP	: False positive
FPR	: False positive rate
G-mean	: G-ortalama
Knn	: K-nearest neighbor / K- en yakın komşuluk
KLOC	: 1000 satır kod sayısı
LOC	: Kod satır sayısı
NTA	: Number the attributes / Özellik sayısı
NNO	: Total number of operations / Toplam işlem sayısı
ROC	: Receiver operating characteristic - İşlem karakteristik eğrisi
SV	: Support vektor / Destek vektörü
SVM	: Support vector machine / Karar destek vektörü
TBN	: Transfer naive Bayes
TN	: True negative
TNR	: True negative rate / Özgüllük
TP	: True positive
TPR	: True positive rate / Hassasiyet

ŞEKİLLER LİSTESİ

TABLOLAR LİSTESİ

ÖZET

Anahtar kelimeler: Hata tahmini, yazılım metrikleri, yazılım kalitesi, makine öğrenmesi

Yazılım hata tahmini, yoğun çaba gerektiren ve yazılım geliştirme maliyetlerini azaltmaya odaklanmış metotların geliştirildiği işlemleri içerir. Veri analizleri için istatistiksel ve makine öğrenmesi metotlarının sıklıkla kullanıldığı bu işlemler hata veri setlerindeki bozukluk veya eksikliklerden kaynaklanan yanlış sonuçları üretebilmektedir. Bunlara ek olarak sınıf dengesizliği (class imbalance) olarak adlandırılan hatalı verilerin sistemin belirli bölgelerinde yoğunlaşmasından kaynaklanan sorunlar da ortaya çıkmaktadır. Tahmin edici modellerin doğruluklarını arttırabilmek için hata veri setlerinin bir ön işlemeden geçmesi gereklidir. Ancak bu şekilde güvenilir veriler üzerinde çalışılabilir. SMOTE, rastgele örnekleme gibi yöntemlerle hata veri setleri üzerinde işlem yapılmaktadır. Bununla beraber hata veri setlerine yönelik özel bir örnekleme tekniği bulunmamaktadır.

Tezin amacı büyük veri setlerinde hızlı çalışabilen ve tekrar eden verileri yüksek doğrulukla tespit edebilen bir ön-işleme algoritması geliştirmektir. Algoritma, metrik türetimi ile ilgili işlemleri kapsar. Bir diğer amaç ise algoritmanın makine öğrenmesi alanında kullanılabilecek hata veri setlerine yönelik özel bir yöntem geliştirmektir. Deney sonuçlarında gözlenen kesinlik değerleri literartüdeki diğer ön-işleme yöntemlerinden üstün olduğu takdirde makine öğrenmesi ikili sınıflandırma veri setlerinde de kullanılabilir. Hata kesinliği yöntemle birlikte arttığı için yazılım geliştirme maliyetlerinin azaltılmasına katkı sağlamaktadır. Veri madenciliği verilerinin temizlenmesi için geliştirilecek yöntemlere algoritmanın yön vermesi beklenmektedir. Yöntem, ANOVA, t-test, ki-kare gibi istatistik tabanlı metotlar kullanarak ikili sınıflandırma verilerinde tekrar eden verilerin ortadan kaldırılmasını sağlayan bir veri temizleme algoritmasını kapsar. Buna ek olarak düşük seviyeli metrik türetiminin öğrenme algoritmalarının başarısına etkisi gözlemlenmiştir.

Kullanılan makine öğrenmesi yöntemleri ve istatistiksel işlemler ile önerilen ön-işleme algoritmasını da içeren bir yazılım çerçevesi C# programlama dili kullanılarak geliştirilmiştir. Bu çerçeve çeşitli formattaki hata veri setleri üzerinde ikili sınıflandırma performans analizlerini ve temel istatistiksel işlemleri de yapabilmektedir. Önerilen yöntem 15 endüstriyel ve 5 açık kaynak olmak üzere toplamda 20 adet yazılım proje veri seti üzerinde denenmiş eğrinin altında kalan alan (auc) ve kesinlik (precision) performans parametrelerinde algoritmanın etkisi dört farklı sınılandırıcı kullanılarak ölçülmüştür. Sonuç olarak HSDD algoritması iki değerlendirme parametresinde açık kaynak kodlu projelerde daha iyi sonuçlar üretmiştir.

A NEW IMPROVED DEFECT PREDICTION FRAMEWORK FOR SOFTWARE DEVELOPMENT USING REPEATED DATA ANALYSIS

SUMMARY

Keywords: Defect prediction, Software Metrics, Software Quality, Machine Learning.

Defect prediction includes processes which develop methods that are required intensive effort and focused on reducing software costs. These processes, which use statistical or machine learning based techniques, may produce misleading results caused by skewness or deficiencies in defect data sets. In addition, a particular issue namely class imbalance is that reveals because of the wrong distribution of defects which are more in a part of the system than the other parts. In order to increase the accuracy of the predictors, defect data sets should be exposed to preprocessing. Merely this way provides a reliable investigation on right data. Some preprocessing methods such as SMOTE and random sampling help practitioners to obtain balanced data. However, there is not any special technique for defect prediction data sets.

The aim of this thesis is to develop a preprocessing algorithm which fast and gives high precision along with large-scale data especially in defect prediction data sets. The algorithm comprises the processes which are related to the deriving low level metrics. Another aim is to make this algorithm more suitable for other machine learning data sets. . If experimental results are more suitable than other preprocessing methods in literature, the method could be used in other binary classification data sets. Due to the increase of the precision in results, software development cost decreases remarkably by this way. The method is expected to led new alternate data mining cleansing methods. If proposed method is applied in industry, it helps to reduce the cost of testing process that accounts for roughly 50 percent of development process. The method comprises the cleansing algorithm using statistical methods such as ANOVA, t-test, chi-square in which the removing process is preceeded. Further, the effects of the deriving low level metrics in the success of learning algorithms were observed.

A new framework including machine learning methods and statistical techniques is proposed in which developed using C# programming language. This framework is able to employ various data sets having different formats. Besides, proposed framework provides analyzing binary classification data and makes base statistical operation. The method was applied on 20 data sets including five open source and AUC and precision performance parameters. HSDD produced good results in open-source projects.

BÖLÜM 1. GİRİŞ

1.1. Motivasyon

Yazılım hata tahmini, yazılım geliştirme ve bakım bütçelerinin iyi planlanmasına yardımcı olan bir dizi aktiviteyi kapsar [1]–[3]. Yazılım boyutlarının artması bu aktiviteleri zorunlu hale getirmektedir. Yazılım boyutları arttıkça, hata tahmini için geliştirilen modeller karmaşıklaşmakta bu da daha çok zaman ve kaynak ihtiyacını beraberinde getirmektedir [4], [5]. Hata tahmin modelleri bu ihtiyaçların planlanmasına yardımcı olmaktadır. Hata tahmin modelleri, yazılım hata tahmin veri setleri oluşturulurken kullanılan metrik standartları dikkate alınarak geliştirilmektedir. Bu standartlardan en çok kullanılanları LOC, McCabe, Halstead, CK, ve OO [6]–[8]. Bununla beraber, sadece belirli standartları kullanmak hata veri setlerinin kalite düzeylerinin güvenilirliği için yeterli değildir [9].

Hata tahmin deneyi bir ikili sınıflandırma problemidir [10]. Nitekim birçok çalışma ikili sınıflandırma algoritmalarının başarıları karşılaştırılarak gerçekleştirilmiştir. Bunlar istatistiksel, analoji tabanlı, sinirsel ağ, destek vektör tabanlı, karar ağacı ve topluluk metotları olabilir. Bununla beraber kullanılan sınıflandırıcının önemi hata tahmin veri seti kalitesi ve diğer süreç metriklerinden daha azdır [11], [12]. Veri setlerindeki gereksiz veya eksik özellikler sınıflandırıcının başarısını azaltır [13].

Hata veri setlerindeki kalite problemleri ihmal edilebilir değildir ve tahmin performansını önemli ölçüde etkiler [14]. Veriler üzeride yapılan örnekleme ve budama gibi işlemler ile bu sorunla mücadele edilmeye çalışılır [15]. Yalnızca örnekleme veya budamadan birinin kullanılması daha farklı sınıflandırma problemlerini beraberinde getirir. Dengesiz veri setleri için sadece budama işlemi başlı başına faydalı verilerin ihmal edilmesi riskini içermektedir. Bu nedenle iyi bir eğitim elde edilmeyebilir. Diğer taraftan örnekleme tekniği büyük veri setlerinde yük getiren, çaba gerektiren işlemdir.

Daha kötüsü, verilerin eğitim süresi artar ve daha fazla bellek gerektirir. Bu iki yöntemin karışımı olan melez bir yöntem çözüm olabilir [16].

Statik kod metrikleri veya süreç metrikleri literatürde sıklıkla tartışılmaktadır [17], [18]. Süreç metrikleri ile geliştirilen tahmin modellerinin ürettiği sonuçlar umut verici iken [19], [20] statik kod metrikleri ile geliştirilen modeller daha durağandır. Buna rağmen statik kod metrikleri hata veri seti kalitesi açısından tam olarak araştırılmış değildir [21]. Statik kod metriklerinin avantajları otomatik ve ucuz toplanmasıdır [22]. Diğer taraftan tanımlanmamış modüllerin ayırt edilebilmesi için karakter sayısı gibi düşük seviyeli metriklerin kullanılması gereklidir [23]. Ayrıca hata veri setleri birbirine benzer örnekleri içermektedir. Bu örnekler tekrar eden veriler olarak adlandırılmakta ve sınıflandırma başarısını olumsuz etkilemektedir. Bu da tekrar eden verileri doğru bir şekilde tespit edip eleyen bir metodun ihtiyacını ortaya çıkarmaktadır [24].

Bu çalışmada, düşük seviyeli metrik türetiminin sınıflandırmaya etkisi araştırılmış ve tekrar eden verileri temizleyen bir yöntem önerilmiştir. Yöntem tera-PROMISE [25] veri deposundan alınan 15 veri seti ve açık kaynak kodlu beş veri seti [25] üzerinde denenmiştir. 15 veri seti Softlab ve Nasa Metric Data Program (Nasa MDP) veri setlerinden oluşmaktadır. Ön-işleme öncesi durumla ön-işleme sonrası durumu karşılaştırmak için Bayes, Naive Bayes, Rastgele Orman (Random Forest) ve J48 algoritmaları ile sınıflandırma başarısı ölçülmüştür. Elde edilen sonuçlar eğrinin altında kalan alan (area under the curve, AUC) ve kesinlik (kesinlik) performans parametreleri ile değerlendirilmiştir. Ek olarak sınıf karmaşıklığı ile hata dağılımları arasındaki ilişki araştırılmıştır.

1.2. Tezin Amacı

Makine öğrenmesinin temel modellerinden denetimli öğrenme detay özelliklerle daha anlamlı olduğundan, tahmin başarısında düşük seviyeli metrik türetiminin sınıflandırıcı başarısını arttırması amaçlanmıştır. Geliştirilen ön-işleme algoritmasının büyük verilerle hızlı çalışması hedeflenmiştir. Algoritmayı oluşturan istatistiksel analizlerin temel işlemleri ortak kod üzerinden çalışacak şekilde düzenlenmiş, böylece veri

analizinde geçen süre kısaltılmaya çalışılmıştır.

Ön-işlemede kullanılacak istatistiksel metotların hata tahmin veri setlerine uygun metotlar olması ve veri seti üzerinde bu analizlere ilişkin hazırlıklara en az ihtiyaç duyan metotların seçilmesi hedeflenmiştir. Örneğin, hata veri setlerinde yer alan metrik alanları genel olarak sayısal verilerden oluşsa da hata-etiket bölümü "true/false" ya da "Y/N" şeklinde yazı ifadeleri içermektedir. Bu ifadeler sayısal olarak "1/0" şeklinde yorumlandığında matematiksel işlemlerde problem oluşturmamaktadır. (Ayrıca seçilen istatistiksel metotlar matematiksel sayı bloklarının ağırlıklandırılarak değişik şekillerde karşılaştırılması için en uygun yolları içerir).

Metrikler arasındaki ilişkilere bakarak ve LOC, Halstead ve Chidamber-Kemerer gibi standartları da inceleyerek düşük seviyeli metrikleri türetmeyi ve bu metrikleri deneysel çalışmadaki veri setlerine değerleriyle birlikte eklemeyi amaçladık. Eklenen metriklerin bazıları yöntem olarak mevcut fakat deneysel verilerde bulunmamakta, bazıları ise literatürdeki standartlarda dahi yer almamaktadır. Örneğin kod karakter sayısı metriği çoklu lineer ilişki çıkarımlarıyla elde edilebilecek düşük seviyeli bir metrik olup metrik standartlarında yer almamaktadır. Bu metrikler, öğrenme başarısına dolayısıyla tahmin başarısına olumlu katkı yapmıştır.

Ön-işleme algoritmasını yazılım hata veri setleri dışındaki diğer veri setleri ile de test etmeyi amaçladık. Bu veriler veri tipi açısından çok değişkenli, sınıflandırmaya uygun ve kategorik verilerdir [26].

Tezin amaçlarını şu şekilde özetleyebiliriz:

i) Yazılım hata veri setlerinde başarılı bir ön-işleme algoritması geliştirmek, ii) NASA MDP ve SOFTLAB veri setlerindeki tekrar eden veri oranlarının açık kaynak kodlu diğer hata veri setlerinde de genel olup olmadığını tespit etmek, iii) düşük seviyeli metrik kullanımının tahminde kesinliğe (precision) etkisini araştırmak ve önerdiğimiz ön-işleme yöntemini veri madenciliğinde sıklıkla çalışılan veri dengesizliği (class imbalance) probleminde kullanılacak bir veri budama yaklaşımı olarak sunmaktır.

1.3. Literatür Taraması

1.3.1. Hata tahmini ile ilgili çalışmalar

Hata tahmin çalışmaları iki yönlüdür: ilki, hata tahmin yatkınlığını tespit etmek için yapılır ikincisi kalan hataları tahmin eder [22]. Song ve arkadaşları [27] tarihsel verileri ve öğrenme şemalarını kullanarak hata yatkınlığını tespit etmeye çalışmışlardır. Çalışma üzerinde çalışılan farklı veri setleri için farklı öğrenme şemaları seçilmesi gerektiğini vurgulamıştır. Yazılım modüllerinin büyüklüklerinin hata yatkınlığı ile ilişkisi de bir başka araştırma konusudur. Büyük ölçekli yazılım sistemlerinde büyük modüller ile karşılaştırıldığında küçük modüllerin daha hata yatkın olduğu tespit edilse de, bu varsayımın genelliğini doğrulamak için tasarım ve kodlama ile ilgili daha detaylı bilgiler veren metriklere ihtiyaç duyulduğu belirtilmiştir [28]. Hata yatkınlığının yazılım geliştirme aşamalarının hangisinde yapılması gerektiği de bir başka araştırma konusudur [29]. Tasarım seviyesinde yapılan hata tespitinin daha faydalı olduğu ancak OO modellerin [30] daha iyi analiz edilmesi gerektiği sonucuna ulaşılmıştır [31]. Hatalı modüllerin daha iyi tespit edilebilmesi için veri seti ve karmaşıklık metriklerinin seçimi de önemlidir [32]. Ancak farklı alanlardaki(domain) veriler için genel bir kural oluşturulamamıştır.

Birçok sınıflandırma algoritmasının başarısı hata tahmini açısından incelenmiştir. Bunlardan en popüler olanları Bayes, Naive Bayes, Rastgele Orman, mantıksal gerileme (logistic regression) ve destek vektör makineleri (support vector machine, SVM) dir [33]–[36]. Kullanılan veri setlerindeki gürültü problemleri, deney koşulları ve verilerin farklı olması bu parametrelere bağlı olarak algoritma başarısında değişkenliğe yol açmaktadır. Dolayısıyla bu algoritmaların üstünlük açısından araştırılması gereksizdir [11], [37].

Hata tahmini yapılırken eğer eğitim için gerekli veriler bir projeden, test için gerekli veriler de başka bir projeden alınıyorsa bu çapraz-proje hata tahmini olarak adlandırılmaktadır [38]. He ve arkadaşlarının 34 veri seti üzerindeki araştırmalarında çapraz-proje hata tahmininde eğitim verilerinin dikkatli seçilmesi gerektiği savunulmuştur. Bu noktada ön-işlemenin önemine dikkat çekmişlerdir [39]. Eğer

rastgele modeller kullanılıyorsa çapraz-proje tahmini iç-proje tahminine göre AUC performans parametresi açısından daha iyi sonuçlar üretmiştir [40]. Ma ve arkadaşları TBN (Transfer Naive Bayes) olarak isimlendirdikleri bir ağırlıklandırma algoritması geliştirmişlerdir [41]. TBN çalışma zamanı açısından umut vericidir. Hedef verilerin maximum ve minimum değerleri alınarak yapılan analizin verinin karakterini tam olarak yansıtmadığı ve veriler daha fazla bilgi alınabilmesi için detaylı metrik araştırmalarına ihtiyaç duyulduğu sonucuna ulaşılmıştır. Kamei ve arkadaşları [42] çapraz-proje modelleri için JIT yaklaşımını sunmuşlardır. Bu çalışmanın temel amacı tasarım aşamasında geliştiriciye hatalar ile ilgili erken geri bildirimler sağlamaktır. 11 açık kaynak kodlu veri seti üzerindeki deneysel çalışmada veri seti seçiminin önemine dikkat çekilmiş ve daha iyi sonuçlar alabilmek için metrik setinin genişletilmesi gerektiği vurgulanmıştır. 8 proje üzerinde yapılan bir başka araştırmada [43] performans parametreleri açısından çapraz-proje hata tahmini daha iyi Geri Çağırma (recall) göstermiştir. Aksine iç-proje tahmininde kesinlik parametresi daha etkilidir. Rahman ve arkadaşları [40] çapraz-proje ile iç-proje hata tahminini dokuz farklı projenin farklı sürümlerini kullanarak karşılaştırmışlardır ve maliyet-hassas analizde çapraz-proje tahmininin iç-proje tahmini kadar iyi sonuçlar üretebildiği tespit edilmiştir.

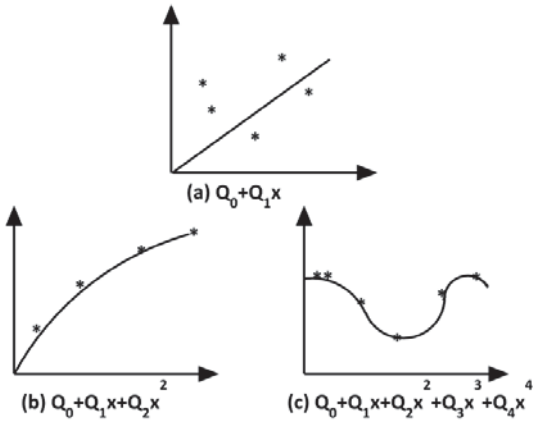
Statik kod metrikleri üzerindeki ilk ses getiren detaylı çalışma Menzies ve arkadaşları tarafından [17] yapılmıştır. Nasa MDP veri setleri üzerinde yapılan deneysel çalışmada McCabe ve Halstead standartları sınıflandırma başarısı açısından karşılaştırılmıştır. Bu çalışmanın önemli sonuçlarından biri daha iyi tahmin sonuçları alabilmek için bu standartların yenilenerek yeni metriklerin türetilmesi gerektiğidir. Aynı veri setlerini kullanan bir başka çalışmada statik kod metrikleri açısından özellik çıkarımı ile ilgili daha fazla araştırmaya ihtiyaç duyulduğu vurgulanmıştır [34]. Karmaşıklık (complexity), eşleme (coupling) ve bağlılık (cohesion) metrikleri yazılım savunmasızlığı açısından incelenmiş ve tahmin işlemlerinin yazılım geliştirme sırasında otomatikleştirilmesi ve süreç metriklerinin de incelenmesi gerektiği belirtilmiştir [44]. Malhotra statik kod metrikleri üzerinden tahmin modellerini karşılaştırmış ve AUC parametresi açsısından makine öğrenmesi modelleri mantıksal gerileme modellerine göre daha üstün gelmiştir [45].

Statik kod metriklerinin dışında süreç metrikleri de hata tahmininde sıklıkla kullanılmaktadır. Rahman ve Devanvu [19] 12 proje üzerinde süreç metrikleri ile statik kod metriklerinin tahmin başarısını AUC parametresi açısından karşılaştırmıştır. Bu çalışmaya göre süreç metrikleri statik kod metriklerine göre daha iyi sonuçlar üretmiştir. Ancak bu sonuç özellikle ticari projeler için veri kalite problemleri dikkate alındığında geçerli olmayabilir. Bu da statik kod metrikleri üzerinde daha detaylı bir ön-işleme sayesinde anlaşılabilir. Foucault ve arkadaşları sahiplik metrikleri ile yazılım kalitesi arasında bir ilişki olduğunu regresyon modelleri ve spearman analizi ile ortaya çıkarmış ancak sonuçların onaylanması için daha geniş bir metrik seti ile deneyin genişletilmesi gerektiğini belirtmişlerdir [46]. Yeni değişen kodlar üzerinde süreç metriklerini kullanarak yapılan tahmin daha iyi sonuçlar üretebilir [20]. Ancak çapraz-proje işlemi ve statik kod metrikleri üzerinde de denenmesi gereklidir.

1.3.2. Örnekleme teknikleri ile ilgili çalışmalar

Hata veri setlerinde sınıflandırmayı olumsuz etkileyen birçok veri kalite problemi vardır. Bu problemlerden biri dengesiz veri setleridir. Dengesiz veri probleminde hatalar sınıfın belli bölümünde yoğunlaşır. Dengesiz verilerle iki çeşit baş etme yöntemi bulunmaktadır: örnek artırma (oversampling) ve örnek azaltma (under sampling) [15]. Örnek arttırma yöntemi örneklerin çoğaltılması esasına dayanırken, örnek azaltma yönteminde örneklerin bir bölümü elenir. Bu işlem arzulanan sınıf dağılımı için yapılmaktadır. Bununla beraber örnek arttırma yöntemi aşırı-uygunluğa neden olurken, örnek azaltma yöntemi bilgi verici örneklerin ihmal edilmesine yol açmaktadır [14]. Pears ve arkadaşları tarafından SMOTE örnek arttırma algoritması Jazz veri deposu üzerinde denenmiş hata kesinliğinde %80 verim elde edilmiştir. Ancak örnek arttırma yöntemi ile türetilen örnekler gerçek dünya verilerin gerçek bir temsili olmayabilir [47]. Sınıf dengesizliği metotları Wang ve Yao [15] tarafından 10 veri seti kullanılarak karşılaştırılmış ve AdaBoost.NC metodu balance, G-mean ve AUC parametreleri açısından en iyi performansı sunmuştur. Hata veri setlerindeki gürültülü örneklerin tespiti için Kim ve arkadaşları [48] CLNI olarak isimlendirdikleri bir tespit algoritması geliştirmişlerdir. Önerilen metot gürültülü örnekleri makul bir

kesinlik ile bulmaktadır. Bu çalışmanın sonuçlarından biri daha az gürültülü örnekler için açık kaynak kodlu sistemlerde hata raporları ile ilgili bilgilerin geliştiriciler tarafından eksiksiz bir şekilde girilmesi gerektiğidir. Ayrıca şu anki çalışmalar gürültülü veriler ile baş etme problemini tam olarak çözmüş değildir. İkili bağımsız değişken (hatalı ve hatalı-değil) yerine çoklu-kategorili tahmin edicilerin kullanımı Shatnawi [49] tarafından araştırılmıştır. Açık kaynak kodlu CK metriklerini barındıran veriler ile test edilen metot yedi sınıflandırıcıda umut verici sonuçlar üretmiştir. Sonuçların geçerliliği için yöntemin dengesiz ve gürültülü verilerin azaltılmasına yönelik etkisinin araştırılması gerektiği vurgulanmıştır. Şekil 1.1 (a)'da özellik sayısı az olduğu için bulunan denklem yanlış hüküm içermektedir. Şekil 1.1 (c)'de ise eğitim eğrisi tüm örnekleri gürültüler ile birlikte yakalamıştır. Bu durum aşırı-uygunluktur. Aşırı-uygunluğu aşabilmek için ağırlıklandırma veya özellik sayısının azaltılması Şekil 1.1 (b)'deki gibi çözüm olabilir.



Şekil 1.1. Aşırı-uygunluk probleminin grafik yardımıyla gösterilmesi, Şekil (a) başarısız öğrenme durumunu, Şekil (c) gürültülü örneklerin de öğrenilmesi (aşırı öğrenme) durumunu, Şekil (b) ise ideal eğridir

1.4. Literatürdeki Eksiklikler

Yazılım parçalarının sınıflandırılması için bileşenlerin hata-yatkın ve hata-yatkın-değil şeklinde belirtilen özellikleri kullanılmaktadır. Bu özellikler belirlenirken daha önceden oluşturulan metrik tablolarından faydalanılmaktadır. Makul düzeyde bir veri kalitesinin olması, başarılı bir hata tahmini yürütebilmek için gereklidir [50], [51]. Fakat yazılım hata verilerinin elde edilmesi oldukça zordur. Ticari yazılım geliştiricileri genellikle hata ölçüm verilerine sahip değildirler. Ayrıca böyle bir hata ölçüm sonucunun genelle

paylaşılması ticari açıdan istenmez. Eğer hata seviyesi ciddi düzeylere ulaştıysa kalite sonuçlarını gizlemek kabul edilebilir fakat bu sonuçlar araştırmacılarla paylaşılırsa gelecek hataların önlenmesi için bir şans doğar.

Açık kaynak kodlu sistemler, araştırmacılar tarafından hata veri setleri oluşturmak için sıklıkla kullanılmaktadır [52], [53], [54]. Geliştiriciler tarafından kullanılan hata-izleme yazılımlarından türetilen hata veri setlerinde çeşitli sorunlarla karşılaşılmaktadır. Bu sorunlardan biri hata bilgilerinin doğru ve tutarlı olup olmadığıdır. Bu açıdan karşılaşılan sorunların temel nedenleri: veri ön-işleme öncesi yetersiz dokümantasyon, makine öğrenme metotlarının uygulanması öncesi verilere yetersiz çaba gösterilmesi ve eksik raporlamadır [55]. Yapılan işlemlere insan müdahalesi gerektiği için bu veri setlerinden doğru bir şekilde hata verilerini elde etmek oldukça zaman alıcı bir görevdir. Proje büyüklükleri ve hata bilgi akış hızları bu görevi olanaksız hale getirmektedir. Hata ile ilgili bilgilerin doğrulanmasından sonra sürüm kontrol mesajlarına girilmesi bu noktada bir çözüm olabilir. Böylece hata verileri daha sağlıklı elde edilmiş olur. Çalışmada önerdiğimiz ön-işleme algoritması hata verilerinin doğrulanmasına yönelik yeni bir çözüm getirmektedir.

Hata tahmin çalışmalarında çoğunlukla tercih edilen ve aynı zamanda projemizde de kullanılacak olan veri setleri NASA ve PROMISE verileridir. Bu veri setleri belirli düzende oluşturulmuş birçok veri setini içermekle birlikte, veri madenciliği algoritmalarını kullanarak yapılan sınıflandırmalar öncesinde yoğun bir ön işleme gerektirmez. Kullanılan metrik kümesi ayrıntılı olarak açıklandığından sistem modüllerinin özellikleri üzerinde araştırma yapmak kolaylaşır. Araştırmacılar NASA metrik programı (MDP) verilerini kullanırken verilerin belirli bir kalite düzeyinde olduğunu varsayarlar. Fakat ikili sınıflandırma sırasında belirli ön işleme aşamalarından geçmeyen veriler hatalı sonuçların çıkmasına sebep olabilir [25]. Bu noktada önerilecek bir ön-işleme algoritması ikili sınıflandırma problemlerindeki veri kalitesinin önemine dikkat çekebilir ve araştırmacıları kullanılan verileri kalite seviyesi noktasında sorgulamaya sevk edebilir.

Hata tahmini ile ilgili çeşitli çalışmalar literatürde mevcuttur. Bunlardan biri Fenton ve

arkadaşlarının çalışmasıdır [56]. İstatistiksel açıdan tahmin yaklaşımı kullanılan çalışmada Bayes modeli kullanılmıştır. Tahmin işlemleri için kullanılan 22 sınıflandırma modeli NASA MDP kullanılarak bir diğer çalışmada incelenmiş, sonuç olarak LogReg, LP gibi lineer modeller benzer sonuçlar üretmiştir [27]. Sınıf dengesizliği üzerine yapılan çalışmalardan birinde [15] 10 veri seti üzerinde 5 farklı sınıf dengesizliği öğrenme metodu karşılaştırılmış ve AdaBoost.NC metodu en iyi sonucu üretmiştir. Fakat sadece C4.5 sınıflandırma metodunun kullanılması daha fazla araştırma yapılması gerektiğini göstermektedir. Geliştiricilerin hata oluşumu üzerindeki etkisi de bir başka araştırma konusudur. Menzies ve Koru 2013 yılında bu konuyla ilgili yaptıkları araştırmada [57] dosyalar üzerinde değişiklik yapan geliştirici sayısının bilinmesinin tahmin için kullanışlı olduğunu, ancak hangi geliştiricinin bu değişikliği yaptığının önemli olmadığını vurgulamışlardır. Bir diğer çalışmada [3] geliştiricilerin sisteme erişim bilgileriyle hata düzeyi arasındaki ilişki araştırmış, otomatik bir hata tespit aracı ile en problemli dosyaların tespit edilmesinin yapılan çalışmaya yön vereceği vurgulanmıştır.

Tarihsel veriler yerine örnek tabanlı hata tahmininin gerçekleştirildiği Li ve arkadaşlarının çalışmasında [3] 5 adet PROMISE veri seti kullanılmıştır. Çok hızlı değişen projelerde tarihsel verilerin kullanımının güçlüğüne dikkat çekilmiş ve sorunun çözümü için ACoForest olarak adlandırılan bir öğrenme metodu geliştirilmiştir. Geliştirilen metot 0.685 F-skor ile Mantıksal gerileme, Naive Bayes, Karar ağacı ve CoForest algoritmalarından daha iyi performans elde etmiştir. Burada PROMISE veri setlerini ve Hall'ın [58] çalışmasındaki gibi benzer sınıflandırıcıları kullanmamızın temel nedeni teorik doğrulama için uygun olmasıdır.

Eğer statik kod özellikleri kullanılarak hata tahmin işlemi yapılmak isteniyorsa, veri madenciliği yöntemlerinin iyi bilinmesi gerekmektedir. Menzies ve arkadaşlarının 2007 yılında basılan yayını makine öğrenmesi tekniklerinin ve performans analizlerinin nasıl yapılması gerektiği ile ilgili en çok atıf alan çalışmalardan biridir [17]. Menzies ve arkadaşları hata tahmin ediciden (predictor) ziyade seçilen özellik kümesinin tahmin başarısını daha çok etkilediğini vurgulamışlardır. 10 proje üzerinde yapılan 10 tekrar sonucunda Naive Bayes(log-filtreleme ile) algoritması J48 algoritmasına göre 71%

sonuçla daha iyi bulunmuştur. Naive Bayes algoritmasını performans değerlendirmede seçilmesinin nedenlerinden biri budur.

Her öğrenme metodu her hata veri seti için uygun değildir. Örneğin Song ve arkadaşları yeni bir hata tahmin çerçevesi geliştirmiş [27] ve bu görüşün doğruluğunu savunmuşlardır. Buna ek olarak çalışmalarının Menzies ve arkadaşlarının [17] çalışmasından farklı yönlerini ortaya koymuşlardır. Bunlardan birisi Menzies ve arkadaşlarının önerdiği çerçevenin sonuçlarının seçilen veri setine göre değişkenlik göstermesidir. Bu açıdan Song'un çerçevesi daha tutarlı gözükmektedir. Ayrıca kullanılan veri setine göre öğrenme şeması seçilmesi gerektiği vurgulanmıştır. Hata raporlarını doğrudan tahmin modeline sokmak doğru olmayabilir. Tahmin öncesi bir ön temizleme tahmin başarısını arttırabilir. Bu bağlamda Kim ve Kim'in çalışmasında [48] iki aşamalı bir tahmin modeli geliştirilmiş, ilk aşamada yetersiz raporlar elenmiş ve 70% oranında doğru tespit yapılmıştır. Bu çalışma da yapılan ön işleme bakımından çalışmamızı desteklemektedir. Ancak raporları sadece yeterlilik bakımından incelememek, buna ek olarak benzer örnekleri içeren veri setlerinin tespit edilip eğitim işlemi öncesinde elenmesi gerektiğini savunuyoruz.

İkili sınıflandırma problemlerinde performans değerlendirmesi için ROC eğrileri ve bu eğrilerden elde edilen AUC, kesinlik, Geri Çağırma değerleri kullanılır [65]. Önerdiğimiz hata tahmin çerçevesindeki sınıflandırma başarım analizlerini bu değerleri kullanarak gerçekleştirdik. Bu işlemleri yaparken temel aldığımız çalışma Davis ve Goadrich'in çalışmasıdır [59]. Önerilen çerçeveyi geliştirirken NASA MDP verilerini kullanmamızın temel nedenlerinden biri farklı modelleme tekniklerine imkân vermesidir. Fakat kaynak kodlarına erişimin olmaması kapsamlı bir araştırma için kısıtlama oluşturur [60]. Bu noktada hata raporlarının el ile incelenmesi gerekebilir. Herzig ve arkadaşları [55] 5 proje üzerindeki deneysel incelemelerinde 39% oranında dosyalarda hata olmadığı halde hatalı olarak işaretlendiğini tespit etmişlerdir. Hata raporlarının daha iyi analiz edilebilmesi için açık kaynak kodlu olması gerektiğini vurgulamışlardır. Hataların yazılımın belirli bir bölümünde yoğunlaşması sınıf dengesizliği olarak adlandırılmaktadır. Sınıf dengesizliği probleminin çözümü için iki sınıflandırıcı önerilmiş, deneysel veriler için NASA ve SOFTLAB kullanılmıştır.

AKPCAC olarak adlandırılan algoritma F-ölçüt (F-measure), Freidman's ve Tukey's [24] testlerinde diğer algoritmaya göre daha iyi sonuç vermiştir.

AUC sınıflandırma modellerinin başarısı, karşılaştırmalarda sıklıkla kullanılmaktadır. Örneğin Lessmann ve arkadaşlarının çalışmasında [11] 20 adet sınıflandırıcı AUC değerleri ile kıyaslanmış ve AUC'nin ayırt edici olduğu sonucuna varılmıştır. Özellikle LogReg, LP gibi lineer modeller benzer sonuçlar üretmiştir. Tahmin başarısını temel alarak yapılan bir değerlendirmenin yetersiz olacağı süreç metriklerinin de değerlendirilmesi buna ek olarak yeni metriklerin tasarlanması gerektiği sonucuna varılmıştır. Buradan yola çıkarak bu çalışmada metrik türetimi konusu ele alınmıştır.

NASA MDP verilerini kullanarak yapılan çalışmalardan birisi de Gray ve arkadaşlarının çalışmasıdır [23]. Veri temizleme üzerine odaklanılan çalışma 13 veri setinden elde edilen metriklerin ikili sınıflandırmaya uygun hale getirilmesi amacıyla belirli özelliklerin silinmesini kapsar. Değerleri belli olmayan metriklere sıfır değeri atanmıştır. Sonuçlardan ilki kullanılan veri setinin genişletilmesi gerektiğidir. Bu sayede tekrar eden verilerin genel olup olmadığı bulunabilir. İkincisi tekrar eden verilerin tespiti için daha düşük seviyeli metriklerin kullanılması, üçüncüsü de sınıflandırmada tekrar eden verilerden kaynaklanan sorunların bulunmasıdır.

Yukarıdaki çalışmaların tamamı kullanılan metoda bakılmaksızın hata tahmin sonuçlarının daha sağlıklı elde edilebilmesi için tahmin verileri üzerinde derin incelemelerin gerektiğini anlatmakta, literatürde sıklıkla kullanılan statik kod metriklerinin yazılım modüllerini daha detaylı anlatan düşük seviyeli metriklerle birleştirilmesini vurgulamaktadır. Ayrıca hata tahmin verilerine özel olarak tasarlanmış bir ön-işleme yöntemine ihtiyaç duyulmaktadır.

Tablo 1.1'de literatür eksikleri görülmektedir. Tablodaki "Çalışma" sütunu çalışmayı yapan kişileri ve çalışma ismini tarihi ile birlikte vermektedir. Çalışmanın deneysel tasarım özelliği "Deneysel tasarım" sütunundan kontrol edilebilmektedir. "Veri kaynağı" sütununda kullanılan deneysel verinin açık kaynak veya endüstriyel veri setlerinden oluştuğu anlaşılmaktadır. Ayrıca bu sütunda deneysel veri seti adedi kaynak

adresi varsa birlikte sunulmuştur. "Sonuç" sütununda çalışmanın bulguları yer almaktadır.

Tüm tablo özetlenecek olursa ilk olarak geçerli bir hata tahmin çalışması için deneysel ortamın gerekli olduğu görülmektedir. Ayrıca NASA MDP veri ambarı gibi deneysel verilerin geçerliliği yüksek olmalıdır. Sonuçları genelleyebilmek için endüstriyel veri setleri ile birlikte açık kaynak kodlu veri setlerinin kullanılması gerekmektedir. Hata tahmin çalışmalarında özellikle veri kalitesi sorunları üzerinde durulduğu ve gürültülü verilere yönelik yeni yöntemlere ihtiyaç duyulduğu anlaşılmaktadır.

1.5. Tezin Bilime Katkısı

Veriler üzerinde işlem yaparken sıklıkla kullanılan iki yöntem örnek-arttırma (oversampling) ve örnek-azaltma (undersampling) dir. Bu yöntemler, kullanılacak verinin büyüklüğüne, nasıl elde edildiğine ve kapsamına bakılarak kullanılmaktadır. Yapılan çalışmalar göstermiştir ki veriler üzerinde bu iki yöntemi de içeren hibrid bir metot geliştirmek en doğru seçimdir. Önerdiğimiz yöntemin özgün değerlerinden ilki metrikler arasındaki ilişkileri ortaya çıkarıp bu ilişkileri kullanarak örnek arttırma işlemini gerçekleştirmesi, diğeri ise tekrar eden veri oranlarını istatistiksel yöntemleri kullanarak bulması ve bu veriler üzerinde gerekli budamaları yapmasıdır. Böylece hem örnek arttırma hem de örnek azaltma işlemlerini içeren yeni bir hibrid yöntem sunmuş oluyoruz.

Çalışmamız seçtiğimiz deney verilerinin NASA MDP ve SOFTLAB verilerinden oluşması veri güvenilirliği ve daha önce yapılan çalışmalarda da sıklıkla kullanılması bakımından bilimsel kalite unsurunu önce çıkarmaktadır. Geliştirdiğimiz algoritma örnek arttırma işlemi için kullanılan SMOTE algoritmasına bir alternatif oluşturabilir. Böylece makine öğrenmesi alanına veri ön-işleme için daha çok tercih edilen bir örnek arttırma metodu kazandırılabilir. Doğal olarak önerilen yöntemin genel geçer olabilmesi için yazılım hata veri setleri dışında başka alanlarda farklı büyüklükteki veri

Tablo 1.1. Literatür eksiklikleri özeti

Çalışma	Deneysel tasarım	Veri kaynağı	Sonuç	Eksiklik
Herzig ve arkadaşları 2013."It's not a bug, it's a feature: how misclassification impacts bug prediction".	Var	Açık kaynak kodlu beş veri seti, http://www.ict.swin.edu.au/research/projects/.	Başarılı sınıflandırma için veri kalitesinin ve raporlamanın önemine dikkat çekilmiş.	Deney geçerliliği yalnızca açık kaynak kodlu verilerde.
D'Ambros ve arkadaşları 2010. "An extensive comparison of bug prediction approaches".	Var	Açık kaynak kodlu beş veri seti, http://bug.inf.usi.ch/.	Tek metrik üzerinden yapılan tahmin verimsizdir. Deneysel veri setleri belirli oranda (%3) gürültü içeermektedir.	Gürültülere yönelik çözüm önerilmemiş.
Fenton ve arkadaşları 2008. "On the effectiveness of early lifecycle defect prediction with Bayesian Nest".	Var	Endüstriyel 31 adet veri seti.	Bayes ağ modeli ile başarılı sonuçlar elde edilmiştir ancak veri toplama problemleri vurgulanmıştır.	Literatürdeki diğer yöntemlerle bir karşılaştırma yok.
Song ve arkadaşları 2011. "A general software defect-proneness prediction framework".	Var	NASA MDP 17 adet veri seti.	Veri özellik seçimi öğrenme algoritmalarının başarısında etkili. Veri seti özelliğine bakılarak öğrenme şeması seçilmeli.	Yalnızca endüstriyel veriler kullanılmış.
Wang ve arkadaşları 2013. "Using class imbalance learning for software defect prediction".	Var	NASA MDP 10 adet veri seti.	Veri dengesizliği öğrenme yöntemlerinin önündeki önemli zorluklardan biridir. Dengesiz veri dağılımında AdaBoost.NC diğer metotlara göre daha iyi sonuçlar üretmiştir.	Yalnızca endüstriyel veriler kullanılmış.
Li ve arkadaşları 2012. "Sample-based software defect prediction with active and semi-supervised learning".	Var	Açık kaynak kodlu 6 veri seti, http://openscience.us/repo/.	Deneysel sonuçlar hem açık kaynak kodlu hem de endüstriyel veri setlerinde test edilmelidir.	Veri azaltma üzerine önerilen bir çözüm yok.
Gray ve arkadaşları 2012. "Reflections on the NASA MDP data sets".	Var	NASA MDP 13 adet veri seti.	Tekrar eden verilerden kaynaklanan sorunların çözümüne yönelik yeni yöntemler geliştirilmeli ve açık kaynak kodlu veri setlerinde test edilmelidir.	Yalnızca endüstriyel veriler kullanılmış.
Hall ve arkadaşları. "A systematic literature review on fault prediction performance in software engineering".	Var	2000-2010 arası yayınlanmış 208 hata tahmin çalışması, ACM Digital Library, IEEEXPLORE, ISI WOS.	Veri seti yeterli büyüklükte olmalı. Veri dengeziliğine bağlı olarak öğrenme başarısı değişmekte.	Dengesiz öğrenmeye yöneilk bir öneri yok.

setleri üzerinde de denenmesi gerekebilir. Diğer taraftan yazılım hata veri setlerinde örnek azaltma ve örnek arttırma operasyonlarının birlikte yapılmaması maliyet ve zaman açısından yazılım üretimlerini olumsuz etkilemektedir. Bu olumsuz etkenler göz önüne alındığında önerdiğimiz algoritmanın olumlu yönde bir iyileştirme yapması beklenmektedir. Song ve arkadaşları [27] kullanılan veri setine göre değişen bir tahmin şeması seçilmesini savunmuşlardı. Önerdiğimiz ön-işleme yöntemi ile Rastgele Orman algortiması gibi bazı sınıflandırıcılarda diğerlerine nispeten daha iyi sonuçların elde edilmesi hedeflenmektedir. Ancak genel olarak bakıldığında sınıflandırıcıya ve veri setine bağlı olmayan ancak hata veri setlerinde daha başarılı bir ön-işleme yöntemin özgünlüğüdür. Buna ek olarak süreç ve statik kod metrikleri sıklıkla kullanılmasına rağmen düşük seviyeli metriklerin ve tekrar eden verilerin tahmin başarısında etkisini ölçen bir çalışma bulunmamaktadır [12][13].

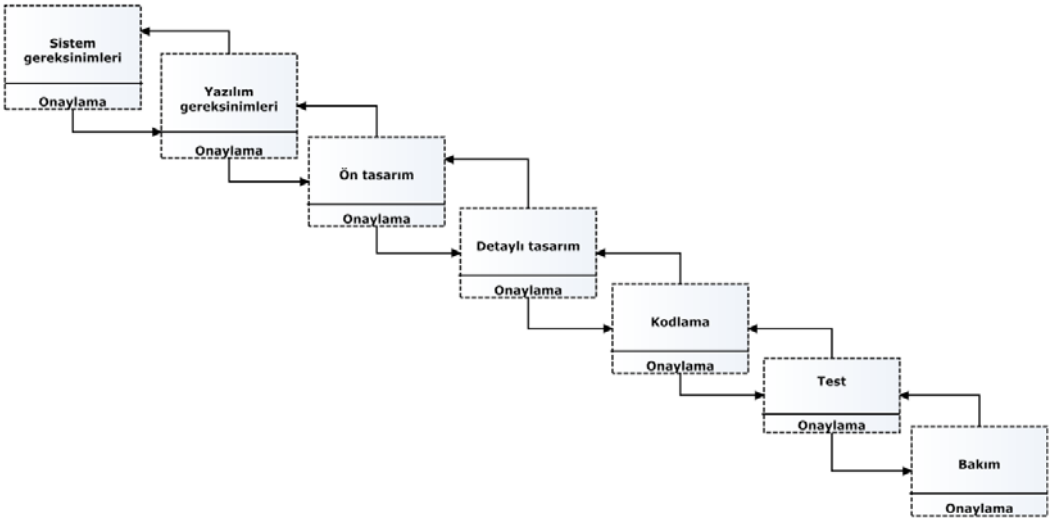
Tez bilime şu katkıları yapmaktadır: 1) Hata tahmin işlemleri için geliştirilen yeni bir çerçeve, 2) Hata veri setlerinde tekrar eden verilerin genel olup olmadığının belirlenmesi, 3) Veri tekrarlarının sınıflandırmadaki etkisinin gözlenmesi, 4) Düşük seviyeli metrik kullanımının tahmin başarısına etkisinin değerlendirilmesi.

1.6. Tez Planı

Tezin geri kalan bölümü şu şekilde organize edilmiştir. Bölüm 2'de yazılım geliştirme metodolojisi hakkında bilgi verilmiş ve yazılım geliştirme modelleri karşılaştırmalı olarak anlatılmıştır. Bölüm 3'te yazılım hata tahmini ve bu alandaki sorunlara yer verilmiştir. Makine öğrenmesi teorisi ve teknikleri Bölüm 4'te yer almaktadır. Bölüm 5'te önerilen hata tahmin çerçevesi ve geliştirilen yazılım detaylandırılmıştır. Deneysel tasarım Bölüm 6'da yer almaktadır. Son olarak Bölüm 7'de bilimsel bulgular ve gelecek çalışmalar tartışılmıştır.

BÖLÜM 2. YAZILIM GELİŞTİRME AŞAMALARI

Yazılım geliştirme, belirli yöntemlerin izlenmesini gerektiren süreçlerdir. Bu yöntemler, geliştirme sürecinin üretkenliğini arttırmaya yönelik çeşitli yaklaşımlar içerir. 1960'lı yıllardan itibaren çeşitli yazılım geliştirme modelleri ortaya çıkmıştır. Zaman içinde bu modeller yazılım teknolojisinin sürekli değişimi ve gelişimine uygun yeni modeller ile değiştirilmiştir. Yazılım geliştirme modelleri belirli standartlar izlenerek geliştirilir. Bu standartlar bu alanda çalışan kişilerin oluşturdukları meslek toplulukları veya diğer iş standart grupları tarafından üretilmektedir. Bunlardan en çok bilinenleri ISO, IEC, IEEE ve ISTQB'dir [61]–[63]. Şekil 2.1'de yazılım geliştirme döngüsünün temel adımları görülmektedir.



Şekil 2.1. Yazılım geliştirme döngüsü

2.1. Yazılım Tasarımı

Yazılım tasarımı, kullanıcı gereksinimlerini uygun hale getiren ve programcıya yazılım kodlaması ve uygulaması sırasında yardımcı olan bir süreçtir [64]. Yazılım gereksinimlerini değerlendirebilmek için gereksinim belirtim dokümanlarına kodlama

ve uygulama aşamalarında ihtiyaç vardır. Bu süreç yazılım terimlerine bakılarak detaylandırılır. Sürecin çıktısı programlama dillerinin uygulanmasında doğrudan kullanılır. Tüm bunlar yazılım yaşam döngüsünün ilk adımını kapsamaktadır. Temel amaç gereksinimleri karşılayabilmektir.

Yazılım tasarımının ortaya çıkardığı sonuçların üç seviyede değerledirilmesi gerekir. İlk seviye olan mimari tasarım seyivesi sistemin en yüksek soyut sürümüdür. Bu seviye, yazılımı birbiriyle ilişkili birçok bileşenin etkileşimde olduğu bir sistem olarak tanımlar. Tasarımcılar bu aşamada sistemin uygun çözümü için fikir edinirler. İkinci seviye yüksek düzeyli tasarımdır. Bu aşamada soyut sistem daha az soyut alt sistemlere bölünür ve modüllerin etkileşimleri vurgulanır. Yüksek düzeyli tasarımda, sistemin modüller biçiminde nasıl uygulanacağına odaklanılır. Modüler yapının her birinin diğeri ile etkileşimi ve modül ilişkileri tanımlanır. Son aşama olan detaylı tasarımda ilk aşamadaki sistem ve alt sistem uygulanmaya çalışılır. Modüller ve uygulanma biçimleri detaylandırılır. Bu aşamada modüllerin mantıksal yapısı ve arayüzleri tanımlanır.

Yazılım sisteminin çoklu ve ayrık modüllere bölünmesi önemli bir işlemdir. Bu işlemde görevlerin birbirinden bağımsız çalışması beklenir. Modüller tüm sistemin temel yapıları olarak çalışabilir. Buradaki ayrık çalışmadan kastedilen tasarımcıların tüm modülleri birbirinden bağımsız derlenebilir ve çalışabilir şekilde tasarlamasıdır. Modüler tasarım 'böl ve yönet' isimli problem çözme stratejisinin kurallarını takip eder [65]. Çünkü modüler tasarımın birçok avantajı bulunmaktadır. Modüler tasarımın avantajları şunlardır:

1. Küçük bileşenlerin bakımı kolaydır;
2. Programlar fonksiyonel açılardan bölünebilir;
3. İstenen seviyede soyutlama programa uydurulabilir;
4. Yüksek bağlılıklı bileşenler yeniden kullanılabilir;
5. Eşzamanlı çalıştırma mümkün hale gelir;
6. Güvenlik açısından istenen yapı elde edilir.

Sıralı çalıştırma yapısında kod komutları programın ilgili bölümü aktif edildiğinde çalışacak şekilde ayarlanır. Birden fazla modül içeren yazılım yapısı için her çalıştırmada modüllerin sadece biri aktif durumdadır. Örneğin bir yazılıma ait modül eğer yazım denetimi yapılacaksa ilgili kelime işlemcisi çalışır.

Modüler yazılımlarda görevler karakteristik özelliklerine bakılarak çeşitli alt modüllere ayrılır. Modüller bazı görevlerin tamamlanması için komut dizileridir. Modüller tek varlık olarak değerlendirilebilir ancak bir diğerini işaret ederler. Modüllerin tasarımı ve birbirleriyle etkileşimini ölçen bazı parametreler vardır: Bunlar bağlılık (cohesion) ve eşleme (coupling) dir.

Bağlılık, modül elemanlarının iç-bağımlılığını ölçen bir parametredir. Bağlılık değerinin yüksek olması program tasarımının iyi olduğu anlamına gelmektedir. Programın küçük alt modüllere bölünmesi rastgele olabilir. Planlanmamış bağlılık programcının kafasını karıştırabilir ve kabul edilmez. Eğer yazılım elemanları mantıksal olarak aynı kategoride ise bir modülde birleştirilir ve buna mantıksal bağlılık adı verilir. Modül elemanları belirli bir zamanda organize edilir ve işlenirse buna geçici bağlılık adı verilir. Modül elemanları belirli bir görevi sıralı şekilde icra ederse buna şekilsel bağlılık denir. Sıralı çalışan modül elemanları aynı veri üzerinde işlem yapıyorsa buna iletişimsel bağlılık denir. Sıralı bağlılıkta modül elemanlarından birinin çıktısı diğerinin girdisi olarak kullanılır. Bağlılığın en yüksek derecesi fonksiyonel bağlılık olup modül elemanları tek bir fonksiyonu gerçekleştirecek şekilde gruplandırılır [66].

Eşleme modüllerin iç bağımlılığını gösteren bir ölçüttür. Eşlemenin düşük olması programın iyi tasarlandığı anlamına gelmektedir. İçerik eşlemede bir modül diğer bir modüle direk olarak erişip bu modül üzerinde değişiklik yapabilir. Aynı veri üzerinde birçok modül işlem yapıyorsa buna genel eşleme adı verilir. Kontrol eşlemesinde bir modül diğer bir modülün fonksiyonuna karar verip o modülün çalışma akışını değiştirebilir. Veri geçişi ile iki modül etkileşimde bulunuyorsa buna veri eşlemesi denir. Eşleme türleri arasında üstünlük açısından karar vermek zordur.

Yazılım tasarım sürecinin çıktısı tasarım dokümanı, simge kodları, detaylı diyagramlar, süreç diyagramları ve fonksiyonel gereksinimlerin tanımlanmasıdır. Yazılım uygulaması aşaması tüm bu çıktılara bağlıdır. Ayrıca bu çıktıların doğrulanması diğer aşamaya geçmenin ön şartıdır. Doğrulama yaklaşımının iyi olması hataların tespit edilip tasarımın güvenilir olmasını sağlar.

2.2. Gereksinim Analizi

Gereksinim, yazılım geliştirme aşamalarındaki tasarım, inşa ve testin gerekli ön değerlendirmelerini kapsar. Gereksinim yönetimi ise planlama, oganizasyon, personel ayrımı ve kontrol aşamalarından oluşur.

Bu aşamalar detaylandırılacak olursa başlangıçta gereksinimlerin dokümanlarının çıkarılması gerekmektedir. Oluşan dokümanlar üzerinden gerekli analizler yapılır. Gereksinimler yazılım geliştirme sürecinin her aşamasında izlenir. Uygulanan modele uymayan gereksinimler değiştirilir veya silinir. Uygulama sırasına bağlı olarak gereksinimlere öncelik verilir. Proje takımı veya paydaşlar tarafından gereksinimler üzerinde anlaşılır. Herhangi bir değişiklik olduğunda tüm paydaşlara gereksinimin durumu bildirilir.

Planlama aşamasının ilk adımı proje gereksinimlerinin toplanması, analizi ve kurulmasıdır. İkinci aşamada kurulan gereksinimlerin kontrollü bir şekilde icra edildiğinden emin olunur. Üçüncü aşamada ise yazılım geliştirme döngüsü boyunca gereksinimlerin takip edilmesi ve müşteriye iletimi sağlanır.

Organizasyon bölümü gereksinimle ilgili işlemlerin etkili ve verimli bir şekilde yürütülmesi için gerekli ortamın hazırlanması ve devam ettirilmesi işlemlerinden oluşur. İstenen kalite düzeyinin yakalanabilmesi için gerekli aktiviteler sürece dahil edilir. Gereksinimler üzerinde yapılacak değişikliklerin kontrollü olması sağlanır.

Personel ayrımında gereksinimle ilgili aktiviteleri yürütecek personel belirlenir, bu personele gerekli eğitimler verilir. Ayrıca gereksinim araçları ve teknikleri personele sağlanarak motive edilir.

Kontrol aşamasında önceki üç aşamada belirtilen işlemlerin planlandığı şekilde yürütülüp yürütülmediğine bakılır, zamanlamanın takibi ve kalitenin istenen düzeyde olması sağlanır.

Risk yönetimi gereksinimlerin onaylanması için gerekli bir süreçtir. Bu süreç gereksinimlerin gerçekten gerekli olup olmadığını belirler. ANSI/EA 632 onaylama işlemini "gereksinim onaylaması" olarak tanımlar [67]. Onaylama sürecinde gereksinimlerle ilgili istenen düzeyde memnuniyetin sağlanması için kullanılabilecek birçok teknik mevcuttur. Bu teknikler simülasyon, test, analiz, gösterim ve inceleme işlemlerini kapsar. Tüm bu işlemlerin belirli bir zaman diliminde tamamlanması gerekir. Teknik performans ölçümü (TPM) gereksinimlerin istenilen zamanda tamamlanmasını takip eder.

2.3. Yazılım Kalitesi

Yazılım kalitesi gereksinimlerin karşılanma düzeyini gösteren bir ölçüttür. ISO, IEC ve IEEE standartlarında kalite tanımı birbirine yakındır [61]. Müşteri ihtiyaç ve beklentilerinin bir üründe, serviste veya bileşendeki karşılanma düzeyi olarak ifade edilir. Kalite faktörleri: fonksiyonel uygunluk, güvenilirlik, performans etkililiği, kullanılabilirlik, güvenlik, sürdürülebilirlik, taşınabilirlik ve uyumluluk olarak sıralanabilir.

Fonskiyonel uygunluk gereksinimlerin beklenen fonksiyonel işlevleri yerine getirme memnuniyeti olarak ifade edilebilir. Müşteri gereksinimleri ile fonksiyonel işlevlerin uyuşması gerekmektedir. Yazılımın fonksiyonlarının aynı zamanda doğru çalışması beklenmektedir.

Güvenilirlik terimi kalite ile birlikte sık kullanılan bir terimdir. Beklenen ve belirtilen servislerin yazılım tarafından ne kadar sıklıkla sunulduğuna bakılır. Bu terim birçok kalite faktörü ile de ilgilidir. Yazılımın doğruluğu yani hatalı olmaması bir güvenilirlik göstergesidir. Kalite faktörlerinden biri performanstır. Burada donanım kaynaklarının yazılım tarafından ne kadar etkili kullanıldığına bakılır. Performansı etkileyen karmaşık teoriler mevcuttur. Kullanılabilirlik de bir kalite faktörüdür. Kullanılabilirlik, kullanıcının yazılımı hangi düzeyde yönetebildiğini belirler.

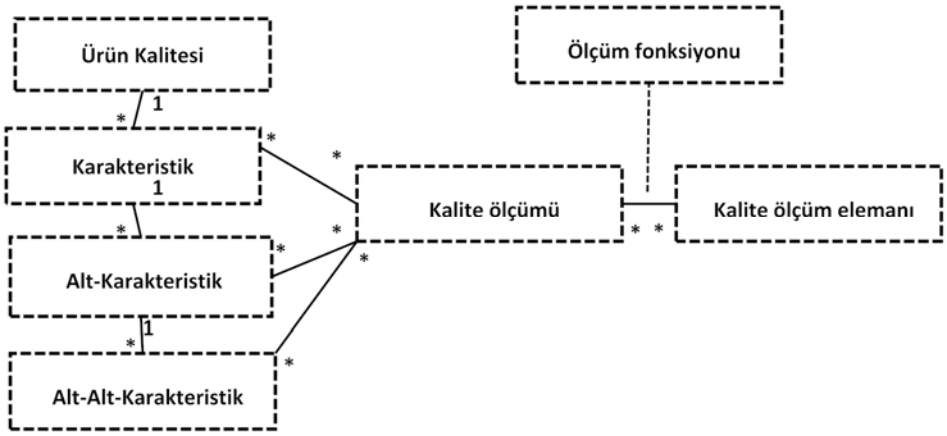
Güvenlik, yazılımın dış saldırılara karşı hazırlık seviyesini gösterir. İnternet bağlantılı tüm yazılımlar için güvenlik düzeyinin yüksek olması beklenir. Donanım ve diğer servisler de güvenlik açısından incelemeye dahil edilmelidir. Bu açıdan tüm sistem için değerlendirilen bir terimdir.

Yazılımlar durağan değildir. Yeni gereksinimler ve değişiklikler ile birlikte yazılım sürdürülebilir olmalıdır. Değişen çevre elemanlarına bağlı olarak da yazılım üzerinde değişiklik yapılabilir. Bu açıdan istenen düzeyde bir sürdürülebilirliğin olması yazılımın yeni sürümlerinde hem kod kalitesi hem de diğer kalite faktörlerinin devamını sağlar.

Yazılımların tek platform üzerinde çalışması beklenmez. Diğer yazılım ve donanım sistemleriyle uyumlu çalışan yazılımlar daha çabuk yaygınlaşır ve kullanıcı çerçevesi genişler. Böylece farklı platformlar üzerinde aynı yazılım sistemine müdahale edilebilir.

Kalite değerlendirmesinin en zor aşaması ölçümdür. Bu ölçümün nasıl yapılması gerektiği ISO/IEC 25020 standartında belirtilmiştir [61]. Temel ölçüm kalite karakteristiğini detaylandırır. Ölçümde her elemana ait bir ölçüm metodu mevcuttur ve bu metotlardan ölçüm bilgileri alınır. Ölçüm elemanlarının nasıl doküman ile detaylandırılacağı bellidir ve her biri erişilebilir fonksiyon, kullanıcı problemleri gibi verileri sunar.

Şekil 2.2'deki modele bakılarak ürün kalitesi, karakteristik, alt-karakteristik, alt-alt-karakteristik aşamalarının sırasıyla ikili ilişkilerinin 1-* olduğu anlaşılmaktadır. Bir ürüne ait birden fazla karakteristik ve alt karakteristik özellikler bulunabilir. Bunların her birine ait kalite ölçümü için bir ölçüm fonksiyonu gereklidir. Ölçüm fonksiyonunun icra edilebilmesi için ayrıca bir ölçüm elemanına ihtiyaç vardır. Kalite ölçüm elemanı her ürün için karakteristik özelliklerine göre kalite raporunu oluşturur. Buradan yazılım kalite düzeyi hakkında fikir sahibi olunabilir.



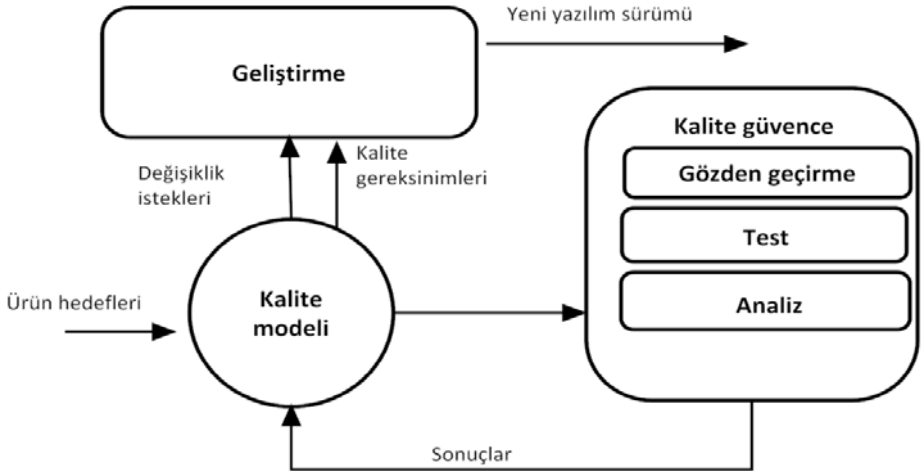
Şekil 2.2. Yazılım kalite ölçüm referans modeli ISO/IEC 2520

Kalite değerlendirmesi kalite ölçümünden sonra yapılan bir işlem olup sadece veri toplama ve ölçüm fonksiyonlarını uygulamadan ibaret değildir. Verilere bir yorum seviyesi eklenir. Yapılan yorumdan sonra ölçüm verilerinin her bir seviyelendirilir. ISO/IEC 25041 standardı geliştiriciler ve bağımsız değerlendiriciler için bir yol haritası sunar [69], [68], [71].

Kalite modelleri çeşitli tanımlama modelini içerir. Tanımlama modelleri yazılım geliştirme süreçlerinin bazı aşamalarında kullanılır. Bu modeller yardımıyla sistemin uygulanması için tavsiyeler ve yaklaşımlar önerilerek yüksek kalite düzeyine ulaşılmaya çalışılır. Şekil 2.3'te yazılım kalite döngüsü görülmektedir. Gözden geçirme, test ve analiz adımları kalite modelini oluşturan adımlardır. Ürün hedeflerine bakılarak ürün-model uyumu denetlenir. Buradan çıkan sonuca göre geliştirme yapılır ve yeni yazılım sürümü ortaya çıkar. Her sürümde adımlar tekrar edilmektedir.

2.4. Yazılım Bakımı

Bakım organizasyonu mevcut yazılımın devamı ve yeni sürümlerin güvenilirliği açısından önemli bir süreçtir. Yazılım geçişleri ve süreçler proje planlarında yer alır. Kontrol listesinde tartışma, açıklama ve etkiler geliştirici tarafından ele alınır. Ürünün teslimi öncesi yapılan işlemler ile ilgili müşteri bilgilendirilir. Bakım kaynaklarında yer alan bilgi transferi ve eğitim geliştirici tarafından sağlanır. Bakım işleminden sorumlu olan kişi ile geliştirici iletişim halinde olmalıdır. Böylece ürün teslimi öncesi ve geçiş sorunları anlaşılabilir. Müşteri memnuniyeti için ürün teslim edilirken yapılan bilgilendirme yeterli olmalıdır. Eğer geliştirici ile bakım personeli arasında bir çatışma



Şekil 2.3. Yazılım kalite döngüsü

olursa bilgi kaybı olabilir. Müşteri yazılımın kısa zamanda ve bütçe sınırlarında teslim edilmesini ister. Bakım organizasyonu iyi yapılmazsa süreçler ve bütçe artar ve müşteri memnuniyeti riske girer.

Müşteri, geliştirici ve bakım personelinin birlikte yapacağı toplantılar bakım planlaması açısından önemlidir. Bakımcı müşteri ile buluşarak geçiş aktivitelerini planlamalıdır. Muhtemel problemleri açıklamalı ve yapılabilecek değişiklikleri göstermelidir. Yeni yazılımda bakım hizmetleri için ayrılan kaynakları tanımlamalıdır.

Her sistem bileşeni üzerinde donanım kaynakları bakımcı tarafından

görüntülenebilmelidir. Her ne kadar bakımcı tarafından tüm işlerin sorumluluğu üstlenilmese de, bakımcı sorumluluğunda olmayan işleri de görüntüleyebilmelidir.

Veriler müşteri tarafından anlaşılır hale getirilmelidir. Raporlama müşteri için yeterli düzeyde olmalıdır. Her müşteri isteği bir yazılım bakım mühendisine atanır ve bu kişi tarafından ilgili olduğu kısım doğrulanmalıdır. Bakım mühendisi yazılım üzerindeki veri değişikliklerini yapabilmelidir. Bunun için bakımcıya gerekli veri izinlerinin sağlanması gerekir. Bakım işlemlerinin kolaylaşması için kullanılan yazılım dili sayısının en aza indirilmesi gerekir. Aksi durumda seçilen programlama dilinin seçilme nedeni belirtilmelir.

2.5. Yazılım Geliştirme Modelleri

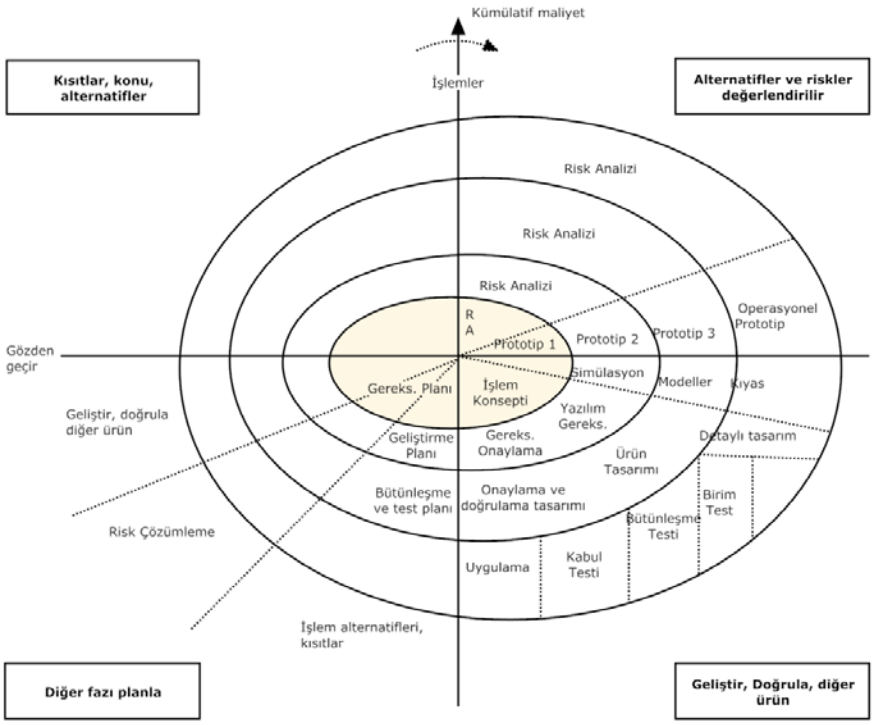
2.5.1. Şelale modeli

1970 yılında önerilen model sıralı adımlardan oluşmaktadır. Bu geliştirme yaklaşımının temel adımları Şekil 2.1'de görülmektedir. Geliştirilecek sistemden beklenenler bu modelde net olarak ortaya konmaktadır. Proje yöneticilerinin işlemleri rahatlıkla takip edebilmelerine olanak sağlayan yöntem diğer geliştirme modellerinin temelini oluşturmaktadır. Şelale modelinin üç temel avantajı mecvuttur. Birincisi kodlama safhasına geçilmeden önce tasarımla hataların yakalanmasını sağlar. İkincisi bakım aşamasında ihtiyaç olacak dokümantasyonu sağlar. Son olarak da gereksinim aşamasından sonra proje maliyetinin tahmin edilmesini sağlar. Bununla beraber gereksinimlerin tasarım seviyesinden önce belirlenip tekrar değiştirilmesinin zor olması geliştirme zamanını arttırmakla beraber çevik geliştirme gibi alternatif yöntemlerin geliştirilmesini sağlamıştır.

2.5.2. Spiral model

Barry Boehm tarafından 1988 yılında geliştirilen model [69], şelale modeli ve hızlı prototipleme gibi yaklaşımların bazı yönlerini birleştirerek temel prensiplerini oluşturmuştur. Şekil 2.4'te spiral modelin adımları görülmektedir. Bu yöntemde risk

analizi üzerinde oldukça yoğunlaşılmıştır. Projenin her ürünü tüm aşamaları tamamlarken spiral oluşturur. Böylelikle ürünler yazılım geliştirmenin erken safhalarında üretilir. Bu model, özellikle büyük ölçekli projeler için uygun olduğu için modeli uygulamak pahalı olabilir. Bu nedenle küçük bütçeli projeler için uygun değildir. Buna ek olarak projenin başarısı büyük oranda risk analiz safhasının iyi düzenlenmesine bağlıdır. Model, daha sonraki yıllarda geliştiricisi tarafından paydaşların kısıtlarını göz önüne alan biçimiyle yenilenmiştir.



Şekil 2.4. Spiral model

2.5.3. Çevik geliştirme

Çevik geliştirmeden bahseden ilk yayın 1974 yılında Edmonds tarafından yazılmıştır [72], [70]. Çevik metotlar 12 temel prensibe dayanmaktadır. Bu prensipler;

1. Müşteri memnuniyeti önceliğini yükseltmek,

2. Geliştirmenin geç safhalarında dahi gereksinimler üzerinde değişiklik yapabilmek, yazılım teslim zamanlarını olabildiğince kısa tutmak, proje boyunca geliştiricilerle yazılım siparişi veren iş sahiplerini birlikte çalıştırmak,

3. Geliştiricilerinin motivasyonunu üst düzeyde tutmak, yazılımı sık sık çeşitli

aşamalarda çalıştırmak,
4. Teknik bilgiye sürekli dikkat etmek,
5. Olabildiğince süreçleri basitleştirmek,
6. Geliştirici ekibin etkililiğini arttırıcı düzenlemeleri geliştirme boyunca devam ettirmektir.

Çevik yöntemlerin farklı yönlerinden biri gereksinimlerin her zaman değiştirilebileceği ve son şeklinin verilmesinin ön görülmediği prensibini savunmasıdır. Büyük ölçekli sistemler için kullanılabilecek yöntemleri içeren çevik metotlar, küçük takımlar ve araçlar ile uygulanabilir. Günümüzün en sık takip edilen geliştirme standartlarını içermesine rağmen yazılım ölçeklerinin sürekli artması ve değişen geliştirme teknolojisi nedeniyle bu yaklaşım ne ilk ne de son olacaktır [71]. Diğer alt bölümlerde çevik yöntem prensiplerini içeren ve günümüzde sık kullanılan üç geliştirme yönteminden bahsedeceğiz.

2.5.3.1. Uç programlama

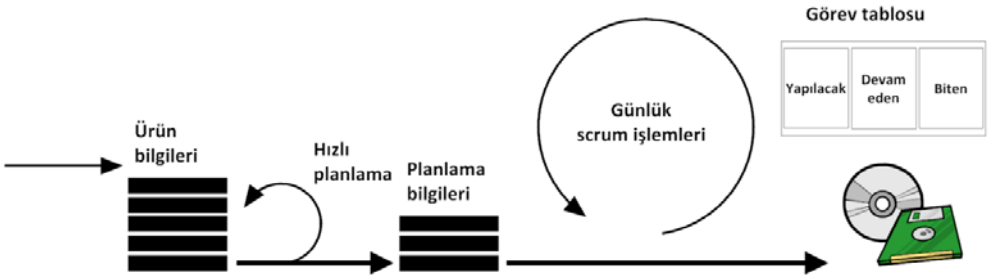
Bu yöntem, küçük geliştirme ekipleri ve sürekli değişen gereksinimlerin olduğu projeler için uygundur. Araştırma, planlama, sürümlerin tekrarı, üretim, bakım ve ölüm olmak üzere altı fazdan oluşmaktadır. Araştırma safhasında yazılımın ilk sürümüne ait gereksinimler belirlenerek geliştiricilerin geliştirme ortamına alışmaları sağlanır. Geliştirmenin önceliklerinin belirlenmesi ve geliştirme takviminin hazırlanması planlama aşamasında gerçekleştirilir.

2.5.3.2. Scrum

Çevik geliştirme metotlarından biri olan Scrum 2004 yılında Schwaber tarafından geliştirilmiştir [72]. Scrum, planlama ve mimari tasarımı olmak üzere iki temel aktivite ile başlar. Planlama safhasında ürün detay listesi oluşturulur. Bu listede tüm ürün gereksinimleri yer alır. Liste takım üyelerinden biri olan ürün sahibi tarafından oluşturulur. Listenin oluşturulma şekli takım yapısına bağlı olarak değişmektedir. Scrum'un önemli özelliklerinden biri ürün listesinin yeni elemanlarla sürekli

güncellenmesidir. Geliştirme takımı tüm sorunlarla ilgili ürün sahibi ile görüşür. Gereksinimlerin önceliklendirilmesi de bu aşamada gerçekleşir. Geliştirme ortamı ve riskler de bu aşamada tanımlanır.

Geliştirme aşaması bir dizi hızlı koşudan (sprint) oluşur. Fonksiyonelliğin bir kısmını yerine getiren tekrar eden yazılım geliştirmesine hızlı koşu denir. Her hızlı koşu, gereksinim analizi, yazılım tasarımı, inşa ve teslim adımlarını içerir. Tüm ürün testleri son aşamada tamamlanır. Scrum geliştirme döngüsü Şekil 2.5'te görülmektedir.

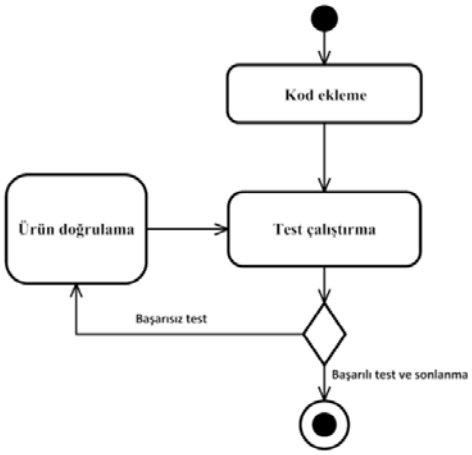


Şekil 2.5. Scrum döngüsü

2.5.3.3. Test-güdümlü geliştirme

Test güdümlü geliştirme, çevik geliştirme esaslarına dayanan kendi içinde tekrar eden aşamaların olduğu, küçük test parçacıkları zincirlerinin çalıştırılması ve yönetimini kapsar [76], [73], [78]. Bu yaklaşımda testlerle ilişkilendirilmedikçe hiçbir kod yazılmaz. Testlerin hazırlanması önceliklidir. Testler bizim ihtiyacımız olan kodları belirler. Şekil 2.6'da test-güdümlü geliştirme döngüsü adımlarını görmekteyiz.

Test işlemlerinin nerede sonlandırılacağı veya test çalıştırma tekrarının adedi ile ilgili bir standart mevcut değildir. Bu geliştiricinin sezgilerine ve geliştirme ortamı kısıtlarına ayrıca teste ayrılan bütçeye göre değişkenlik göstermektedir. Günümüzün geliştirme ortamları (Ör: Visual Studio, Eclipse) yazılan kodlarla ilgili çeşitli test çalıştırma seçenekleri sunmaktadır. Ancak yapılan testin türüne göre bağımsız test araçları da bulunmaktadır. Bunlardan en çok bilinenleri Watir, Selenium, Fitnesse, HP WinRunner, IBM Rational Requisitepro ve SoapUI [80] dir.



Şekil 2.6. Test-güdümlü geliştirme döngüsü

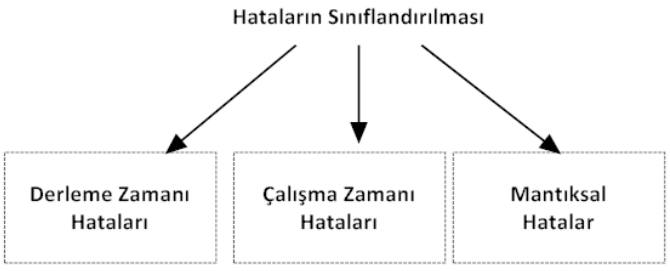
Kullanılan test aracı yapılan testin türüne uygun olarak seçilmektedir. Örneğin grafik arayüzü için Selenium [74], kullanıcı kabul testi için Fitnesse [75], gereksinim analizi için IBM Rational Requisitepro seçilebilir. Yazılım testinde kullanılan teknikler ve yaklaşımlar tezin konusunun dışında olduğu için bu konuları detaylandırmıyoruz. Ancak yazılım hata tahmini test bütçesinin yapılandırılmasına yardımcı olduğu için bu konu kapsamında daha özel bir çalışma alanıdır. Bölüm 3'te yazılım tahmini ile ilgili teorik bilgiler anlatılacaktır.

BÖLÜM 3. YAZILIM HATA TAHMİNİ

Yazılım hata tahmini bir ikili sınıflandırma problemidir. Tahmin deneyinde test sırasında bulunacak hata adedi önceden saptanmaya çalışılır. Genelde sınıflar "hatalı" ve "hatalı-değil" şeklinde önceden etiketlendirilir. Bu işlem bir ön-işlemeyi gerektirmektedir. Sınıflandırmanın yanında gerileme deneyleri de tahmin için kullanılmaktadır. Sınıflandırma işleminde yazılım birimleri kategorilere veya sınıflara ayrılmaktadır. Bu kategori genelde ikili olmaktadır. Tüm yazılım birimlerine ait bir haritalama yazılım sisteminin yapısı hakkında bilgi verici olacaktır [76]. Hata adetleri ile ilgili bir tahmin yapılabileceği gibi hata yatkın yazılım bölümlerinin sıralandırılmasına yönelik deneyler de yapılabilir.

3.1. Yazılım Hatası

Şekil 3.1'de yazılım hatalarının sınıflandırılması görülmektedir. Derleme hataları, derleme zamanında ortaya çıkan geliştiricinin yanlış kodlamasıyla ilgili hatalardır. Bu hatalar yanlış noktalama işareti kullanılması ile ilgili olabileceği gibi, bir kelimenin yanlış yazımından da kaynaklanabilir. Çalışma zamanı hataları, programın çalışma esansında imkansız bir işlemi yapmaya çalışmasından kaynaklanan hatalardır (ör: sıfıra bölme hatası). Mantıksal hatalar, programın derlenip çalışma sırasında hata vermemesi ancak istenen sonucu üretmemesidir.



Şekil 3.1. Yazılım hatalarının sınıflandırılması

Yazılım kalitesi kapsamında anlatılan dört tip hata mevcuttur. Bunlar "derleme hatası (error), hata (defect), kusur (fault) ve böcek (bug)" olarak adlandırılmaktadır. Yazılım hatası, yazılımın iç ya da dış yapısında meydana gelen problemleri adlandırmak için kullanılmaktadır. Hatalarla ilgili standartlar IEEE 610 [77] ve ISTQB [78] terimler sözlüğünde bulunmaktadır.

Yazılım programındaki yanlış bir tanım yazılım kusuru demektir. Bunların dışında yazılımda davranışsal hatalar (failure) da meydana gelebilmektedir. Hata, usur, böcek ve işlevsel hatalar tezin kapsamında değerlendirilebilecek hatalardır.

Yazılım hata tahmininde geçen hata kavramı derleme zamanında ortaya çıkan hatalar değildir. Burada kast edilen çalışma zamanında ortaya çıkan ve kullanıcının gereksinimlerinin karşılanamaması veya fonksiyonel işlevlerden bazılarının düzgün çalışmamasıdır [79]. Bu nedenle kodlama stili ile ilgili hatalar bu kapsam dışındadır. Literatüde "hata (defect), kusur (fault), böcek (bug)" terimleri birbirlerinin yerine sıklıkla kullanılmaktadır [80]. Yazılımlardaki hatalı bölümler sınıflara, modüllere veya bir yazılım paketine ait olabilmektedir. Yazılıma ait sürümlerde çıkan hatalar izlenmekte ve buna karşı önleme yöntemleri geliştirilmektedir.

Yazılıma ait geçmiş sürümlerde çıkan hata adedini kaydetmek ve değerlendirmek yazılım kalitesi hakkında bilgi verebilir. Ancak sadece hata adedi bazında yapılan ölçüm sağlıklı olmayabilir. Hatanın önceliği, ciddiyeti ve daha önceki çıkış nedenlerinin de yer aldığı bir ölçüm daha bilgi verici olacaktır. Ayrıca hatanın bulunduğu yazılım modülüne ayrılan geliştirme zamanı, buradaki kod bloklarının karmaşıklıkları gibi değerler hata ciddiyetinin değerlendirilmesinde kullanılmalıdır.

3.2. Yazılım Metrikleri

Bir yazılım sisteminin veya sürecinin bazı özelliklerinin ölçülmesini belirleyen standarda yazılım metriği denir [81]. Statik kod ve süreç metrikleri olmak üzere iki tip yazılım metriği bulunmaktadır. Süreç metrikleri zaman, maliyet, geliştirici sayısı gibi özellikleri içerir. Statik kod metrikleri ise kaynak kod ile ilgili derleme zamanında

çıkarılan bilgileri içerir.

3.2.1. Statik kod metrikleri

Yazılımlardaki fiziksel kod adedi SLOC veya LOC ile ifade edilmektedir [82]. Programlamanın sürdürülebilirliği için kullanılması gereken bir metriktir. Metin bazında hesaplanan bu metriğin yanında aşağıdaki gibi mantıksal kod LLOC da hesaplanabilmektedir.

```
for (i = 50; i > 1; i--) printf("hesap sonuc:"+i); /* SLOC:1, LLOC:2 */
```

Statik kod metrikleri ile ilgili ilk standart McCabe tarafından 1976 yılında geliştirilmiştir [6]. Bu standart cyclomatic karmaşıklık (cyclomatic complexity) olarak adlandırılmakta ve bir programın lineer-bağımlı yollarının sayısını ölçmektedir. Bu terim program karmaşıklığı (program complexity) veya McCabe's complexity olarak da kullanılmaktadır. Hesaplama programın düğümler ve bu düğümlerin bağlantıları şeklinde çizilmesiyle hesaplanabilmektedir. Eğer bir programın metotlarının karmaşıklığı yüksekse, bu programın test edilmesi de o ölçüde zordur. Dolayısıyla bu durum uygulamanın güvenilirliğini olumsuz olarak etkilemektedir. Yapılacak bilimsel bir ölçümle yazılım metotları yeniden düzenlenerek bu sorun aşılabilir. Programın karmaşıklığı ölçülürken dallanmalar karmaşıklık hesabına dahil edilir. Bu dallanmalar:

1. if tanımları
2. ve || gibi ifadeler
3. for döngü tanımları
4. while döngü tanımlarıdır.

McCabe karmaşıklığı her fonksiyonun, her atomik durumun ve her "case" bloğunu karmaşıklığını 1 kabul eder. Buna ek olarak yazılan kodlar da ayrıca hesaplanır. Karmaşıklık *V(G)* ile ifade edilir. Eğer bu değer 10'dan büyükse program fonksiyonlarının bölünmeye ihtiyacı var demektir. Böylece program daha iyi anlaşılmış ve test edebilir hale gelmiş olur. Karmaşıklık aynı zamanda test durumlarının maliyeti

olarak da yorumlanabilir. Şekil 3.2'de kod üzerinden *V(G)* hesaplama örneği görmekteyiz.

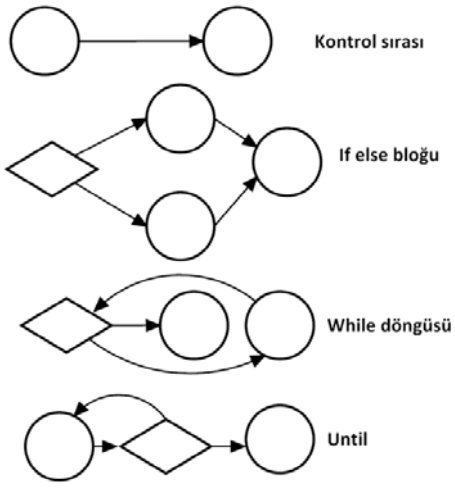
```
... // başlangıç: V(G) = 1

// +2 koşul, V(G) = 3:
if ((i > 10) || (i < 20)) {

 System.out.println("hesaplama!");
//+1 koşul, V(G)=4:
if(i*(-5)<-30)
break;
 // +3 koşul, V(G) = 7:
 while ((i > 0) || ((i > 50) && (i < 90))) {
  ...
 }
}
switch(x) {
 case 1: // +1, V(G) = 8
  break;
 case 3: // +1, V(G) = 9
  break;
 default:
  throw new RuntimeException("x  degeri= " + x);
}
```

Şekil 3.2. Kod üzerinden V(G) hesaplama

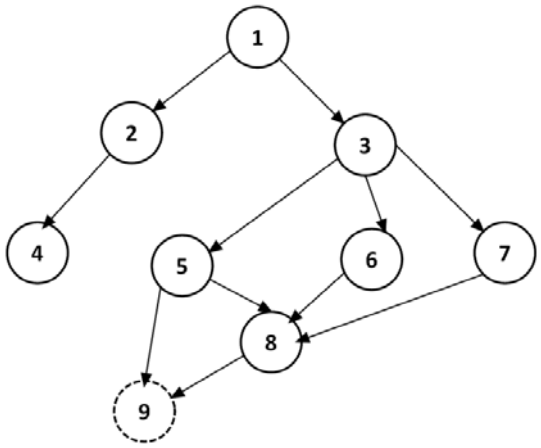
McCabe karmaşıklığı düğüm grafikleri üzerinden hesaplandığında *V(G)= e-n+2p* formülü yardımıyla bulunur. e grafikteki kenar sayısı, n düğüm sayısı ve p bağlantısız düğüm sayısıdır. *V(G)* için kullanılan grafik simgelerinden bazıları Şekil 3.3'te verilmiştir.



Şekil 3.3. Bazı kod ifadelerinin düğüm-kenar gösterimleri

Bu grafik gösterimler kullanılarak Şekil 3.4'teki düğüm-kenar grafiği örneği verilmiştir. Burada kenar sayısı oklarla simgelenmiş olup *e*=11 dir. Düğüm sayısı *n*=9 dur.

Bağlantısız düğümler kesikli çizgili daire ile simgelenmiş olup p=1 dir. Bu değerler kullanılarak $V(G)=11-9+2*1=4$ olarak bulunur. V(G) hesaplama örneği Şekil 3.4'te verilmiştir.



Şekil 3.4. V(G) hesaplama için düğüm-kenar örneği

Statik kod metrikleri için geliştirilen eski standartlardan olup güncelliğini günümüzde de koruyan standartlardan biri Halstead metrikleridir [83]. Temel olarak bu metrikler programlama için harcanan çabayı hesaplamaya yöneliktir. Program içindeki özgün operatör ve toplam operatör sayısı ile özgün işlenen ve toplam işlenen sayısı sırasıyla n_1, N_1, n_2 *ve* N_2 şeklinde ifade edilir. n_1 ve n_2 toplamı *n ile ifade edilir* ($n= n_1+n_2$). N_1 *ve* N_2 toplamı N ile simgelenir. N program uzunluğudur. Bu değerler kullanılarak program hacmi $V=N*log_2(n)$ formülü yardımıyla bulunur. Genel program uzunluğu $L=V*/N$ ile hesaplanır. Programın zorluğu $D=1/L$ ile hesaplanmaktadır. * işareti potansiyel eleman olduğunu göstermektedir. $V' = (n^*_1 + n^*_2) log_2 (n^*_1 + n^*_2)$ denklemi ve $L' = 1/D$ denkleminden elde edilen değerler ile program zekası $I, L'*V'$ ile hesaplanmaktadır. Program için harcanan çaba $E, V/L$ dir. Programın yazılması için gerekli zaman $T'=(n_1N_2(n_1log_2n_1 + n_2log_2n_2) log_2n) / 2n_2S$ dir. S değeri $5<=S<=20$ aralığında seçilmelidir. Örneğin tera-promise veri ambarındaki proje metriklerinde $2n_2S$ 18 alınmıştır. Tablo 3.1'de Halstead metriklerinin özeti görülmektedir. Halstead metrikleri programın derinlemesine bir analizine ihtiyaç duymadan çıkarılabilecek metriklerdir.

Programın kalitesinin basitçe hesaplaması avantajlarındandır. Ancak programın tüm koduna ihtiyaç duyması ve tahmin modeli olarak kullanılamaması dezavantajlarıdır.

McCabe ile karşılaştırdığımızda, McCabe'nin tasarım seviyesinde daha kullanılabilir olduğunu görmekteyiz.

Tablo 3.1. Halstead metrikleri

Metrik	**Açıklama**
v	Hacim
l	Program uzunluğu
d	Program zorluğu
i	Program zekası
e	Harcanan çaba
b	Beklenen hata adedi
t	Geliştirme zamanı
loCode	Kod satır sayısı
loComment	Yorum satır sayısı
loBlank	Boş satır sayısı

Halstead metriklerine göre McCabe metirkleri programın daha erken safhalarında hesaplanabilmektedir. Bu durum test işlemlerini kolaylaştırmaktadır [84].

Nesne yönelimli programlamanın elemanları alınarak geliştirilen metrik tablolarından biri Chidamber ve Kemerer (C&K) nesne yönelimli metrik tablosudur. Altı adet orijinal metrik grubu bu tabloda yer almaktadır [7]. Bunlar ortalama metot sayısı (WMC, weighted methods per class), DIT (miras derinliği, depth of inheritance), NOC (çocuk sayısı, number of children), CBO (nesneler arası eşleme sayısı, coupling between objects), RFC (çalışabilir fonksiyon sayısı, response for class) ve LCOM (bağlılık eksikliği, lack of cohesion) dur. WMC bir sınıf içindeki metot adedini tanımlar. Bu değerin olabildiğince küçük olması gerekmektedir. Bu sayede çıkabilecek hata adedi düşürülmüş olur. WMC değerinin iyi olabilmesi için sınıflardaki metot sayısını 20, 50 gibi değerlerle sınırlamak bir çözüm olabilir. İkinci alternatif ise sınıfların %10 gibi bir bölümünde en az 24 adet sınıf kullanmaktır. DIT metriği herhangi bir sınıfa ait maximum miras derinliğini belirtir. Daha derin sınıfların kullanılması programı daha karmaşık hale getirir. Diğer taraftan metotların yeniden kullanımı kolaylaşır. DIT değerinin büyük olması daha fazla hata olması anlamına

gelmektedir [85]. Bir sınıfa ait alt sınıf sayısı NOC ile ifade edilir. NOC değerinin yüksek olması yeniden kullanımın yüksekliğine, temel sınıfın iyi test edilmesi gerektiğine işaret eder. NOC ve WMC değerlerinin yüksek olması kötü bir tasarım olduğunun göstergesidir. CBO bir sınıf içindeki metot veya özelliklerin başka bir sınıfta kullanılmasıdır. Bu değerin yüksek olması bakım ve testin o sistem için zor olduğu anlamına gelir. Sınıf içindeki metotların çağrılma adedi RFC olarak tanımlanır. Fonksiyon çağrılma miktarına bakılmaksızın her fonksiyon en az bir defa çağrıldıysa bu rakam 1 olarak sayılır. Bir sınıf içinden ne kadar fazla metot çağrılırsa yani RFC ne kadar büyük olursa o ölçüde sınıf karmaşıklaşır [86]. LCOM metriği C&K metrikleri arasındaki en tartışmalı olanıdır. Herhangi bir değişken paylaşmayan fonksiyon adedi bu metrik ile belirlenir. Bu değerin düşük olması hata çıkma ihtimalini arttırmaktadır çünkü bununla beraber karmaşıklık artar [87].

1994 yılında Lorenz ve Kidd tarafından nesne yönelimli tasarım için yeni metrikler önerilmiştir [8]. Bu metrikler C&K metriklerine göre daha sayılabilir metriklerdir. Elde edilmesi kolay olan bu metriklerin kullanışlılığı tartışmalıdır. Çünkü sistem mimarisi hakkında derinlemesine bir analiz sağlamazlar. Bir sınıf içindeki genel erişime açık metot sayısını veren metrik bu standartlarda PM olarak adlandırılmıştır. Herhangi bir sınıfın oluşturulması için gerekli işi tahmin edebilmek açısından önemli bir metriktir. Erişim türüne bakılmaksızın tüm metotların sayısı NM (number of methods) olarak adlandırılmaktadır. Sınıf içindeki genel erişimli değişken sayısı NPV (number of public variables), erişim türüne bakılmaksızın tüm değişkenlerin sayısı ise NV (number of variables) olarak adlandırılır. Bir alt sınıfta miras alınan fonksiyon sayısı NMI (number of methods inherited) ile isimlendirilir. Bu standartlarda yer alan sınıf büyüklüğünü ölçen metrik ise CS (class size)'dir. Bir metotta gönderilen mesaj sayısı ise NOC ile ifade edilir. NOC metriği argümanlı ve argümansız iki tip mesajı kaydetmektedir.

3.2.2. Hata yönetim sistemleri

Hataların yerlerinin tespit edilmesi ve ilgili ölçümleri statik kod metriklerinin elde edilmesine göre daha zordur. Çünkü buradaki ölçümler kişi odaklıdır. Sistemin kullanıcısı olan geliştirici, test ekibi veya son kullanıcı hata bilgilendirmesini yapabilir.

Hata ile karşılaşıldığında bu durum kaydedilir. Bu kayıt, hata ile ilgili ciddiyet, öncelik, yazılım modülü, durum gibi bilgileri içermektedir.

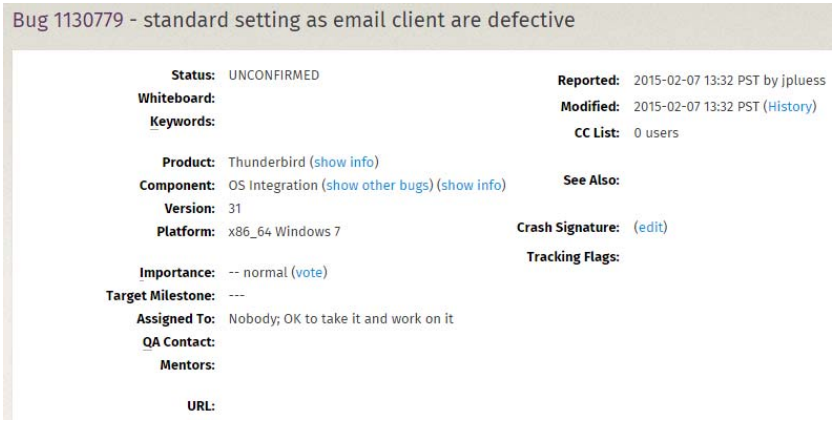
Değişiklik kontrol sistemlerini kullanarak hata yerini tespit etmek mümkündür. Bu sistemler (ör:cvs) değişikliği kimin, ne zaman ve neden yaptığı gibi soruların cevabını içerir [88]. Böyle sistemler hata hakkında kullanıcının yorum yapabilmesine olanak sağlar ki diğer kullanıcılar bu duruma bir çözüm getirebilirler. Ayrıca hataların kişilere atanması, çözülen hataların arşivi ve çözülmeyen hataların nedenlerinin araştırılması bu sistemlerle kolaylaşır. Metin bazında bilgiler içeren yorum mesajlarını araştırıp ilgili hataları daha iyi analiz edebilmek için SZZ algoritması geliştirilmiştir [89]. Algoritmanın ismi geliştiricilerinin isimlerinin baş harflerinden oluşmaktadır. Bu algoritma kod değişikliklerini tarayarak hataya neden olabilecek değişiklikleri kaydetmektedir. Bu algortimaya alternatif algoritmalar geliştirilse de hata ile ilgili önceki değişikliklerle ve son değişikle ilgili farkları belirleyen kesin bir sınıflandırma yapmak oldukça zordur [90], [91]. Bu zorluğu aşabilmek için hata mesajlarına yönelik gürültü filtreleme yöntemleri geliştirilebilir.

Hata yönetimine yönelik çeşitli araçlar mevcuttur. Bunlardan en çok kullanılanları Bugzilla ve ITracker'dır [92]. Bugzilla başlangıçta sadece Mozilla için geliştirilen fakat daha sonra çeşitli yazılım projesi geliştiren ekiplerin kullandığı bir hata izleme sistemidir. Kullanıcı dokümantasyonunun yeterli olması ve kullanım kolaylığı tercih edilmesinde etkendir. Veri tabanından bağımsız bir başka hata izleme sistemi ITacker'dır. J2EE uygulaması olan bu sistem Bugzilla'ya benzemesine rağmen platform-bağımsızdır. Şekil 3.5'te Bugzilla'dan alınan bir hata raporuna ait detaylar görülmektedir.

3.3. Hata Veri Ambarları

Açık kaynak kodlu projelere ait metrik verilerini içeren veri ambarlarından biri Marco ve arkadaşları tarafından oluşturulmuştur [93]. Eclipse JDT Core, Eclipse PDE, Equinox, Lucene ve Mylyn açık kaynaklı projelere ait 18 farklı veri seti grubu bulunmaktadır. Bu veri setleri hataların ciddiyetine ve önceliğine göre gruplananlar,

C&K metrikleri içerenler ve diğer nesne yönelimli olan metrikleri içermektedir.



Şekil 3.5. Bugzilla hata detayları

Yazılım mühendisliği veri ambarlarının en büyüklerinden biri tera-promise veri ambarıdır [26]. Veri setleri kullanım amaçlarına göre gruplandırılmışlardır. Maliyet analizi, gereksinim analizi, hata analizi, hata raporları ve kod analizi ile ilgili veri setleri burada yer almaktadır. Bu veri ambarının en önemli özelliklerinden biri çeşitli hata veri seti gruplarının beraber yer almasıdır. Bunlardan biri "ar" kısaltmalı hata veri setleridir. Bu veri setleri Boğaziçi Üniversitesi yazılım araştırma laboratuvarında oluşturulan veri setleridir [94].

.

BÖLÜM 4. MAKİNE ÖĞRENMESİ VE İSTATİSTİKSEL YÖNTEMLER

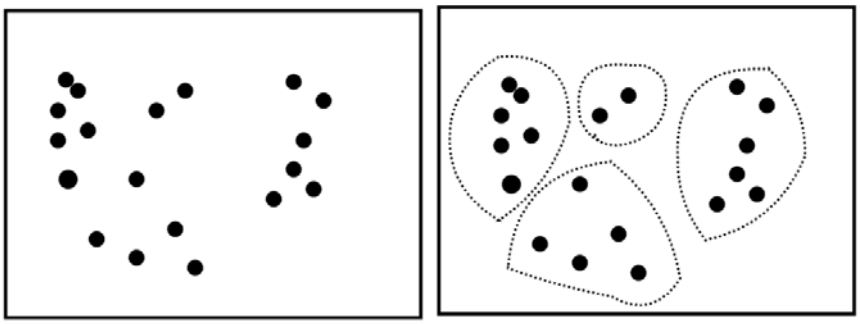
4.1. Teori

Makine öğrenmesi, bilgisayarların öğrenmelerine yönelik yöntemleri içeren, veri madenciliği ile oldukça ilgili geniş bir bilgisayar bilimi alanıdır [95]. Makine öğrenmesi, verilerden modellerin çıkarılmasını veya geçmiş verilerden gelecek verilere ait öngörüde bulunma işlevlerini de içerir. Makine öğrenmesinin popüler alanlarından biri denetimli öğrenmedir. Denetimli öğrenmede geçmiş bilgi ve olaylardan gelecek bilgi ve olaylar tahmin edilir.

Eğer makine öğrenmesi metotları büyük veri tabanlarına uygulanıyorsa veri madenciliği yapılıyor demektir. Burada verilere ait modeller bulunmaya çalışılır. Ancak makine öğrenmesi yöntemleri ile büyük verilere ait modelleri tamamen tanımlamak ve sınırlamak doğru değildir. Amaç yaklaşık ve faydalı çıkarımlar yapabilmektir. Bu sayede geleceğe dönük planlamalar yapılabilir. Makine öğrenmesi sadece büyük veriler ile ilgilenmez. Yapay zekanın bir parçası olduğu için değişen deney ortamlarında öğrenebilir. Makine öğrenmesi özellikle belirli tahmin modelleri için iyi tasarlanmış algoritmalara ihtiyaç duyar. Bu algoritmalar bir çıkarım yaptığı için istatistiksel modeller temel alınarak geliştirilir [96].

4.2. Denetimsiz Öğrenme

Denetimsiz öğrenmede giriş uzayındaki verilerin yapısına bakılarak tüm verilerin istatistiksel yapısı anlaşılmaya çalışılır. Bu çabanın sonunda belirli gruplar veya kümeler oluşturulur. Veri madenciliğini kullanan geniş bir alanda yer bulan denetimsiz öğrenme tekniklerinden kümeleme (clustering) en sık kullanılan tekniklerdendir [97].



Şekil 4.1. Kümeleme

Şekil 4.1'de denetimsiz öğrenme yöntemlerinden kümeleme örneği gösterilmiştir. Verilen veri noktalarından kümelerin oluşturulması ilk aşamadır. İkinci aşamada elde edilen kümelerin başarımlarının değerlendirilmesi gereklidir. Kümelemenin uygun olduğundan ve ne kadar kümeye ihtiyaç duyulduğundan nasıl emin olabiliriz. Bu soruların cevabı tam olarak denetimsiz öğrenmede verilmiş değildir. Bununla beraber verilerin özelliklerine göre kümelerin adetlerini ve belirlenme şeklini araştıran yöntemler geliştirilmeye devam etmektedir. Kümelemenin temel metriklerinden biri öklit uzaklıktır $d(x, y)$, burada $x \in R^n$. Uzaklık metriğinin bazı durumları sağlaması gerekmektedir bunlar: $d(x, y)= d(y, z)$; $d(x, y)>0$; $d(x, y)=0$ eğer $x=y$; $d(x, y)< d(x, z)+ d(z, y)$. Bu eşitlikler kontrol edilerek denetimsiz öğrenmede kümeleme gerçekleştirilir.

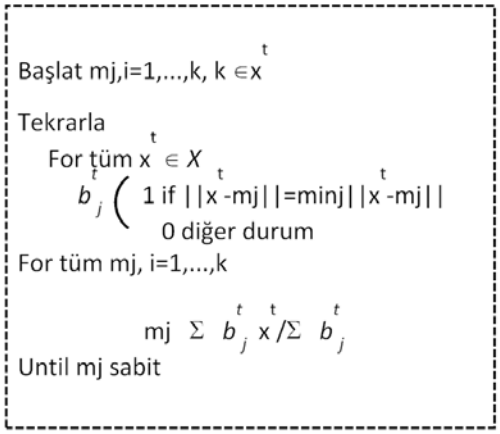
4.2.1. K-means algoritması

K-means terimi ilk olarak James MacQueen tarafından kullanılmıştır [98]. K-means algoritması bilinen kümeleme problemlerini çözmek için geliştirilmiş basit bir öğrenme algoritması olarak tanımlanabilir. Algoritma belirli bir veri seti üzerinden çalışmaktadır. Kümeleme işlemi için daha önceden tanımlanmış küme merkezleri kullanılır. Geçerli bir kümeleme elde edebilmek için küme merkezleri olabildiğince birbirinden uzak seçilmelidir. Diğer adımda her veri noktası en yakın merkez ile ilişkilendirilir. Bu iki adım merkezler sabit kalıncaya kadar tekrarlanır. Algoritma adımları aşağıdaki şekilde özetlenebilir:

1. k veri noktası ilk merkez olarak belirlenir,
2. Tüm noktalar en yakın merkeze atanır,

3. Yeni kümelere göre merkezler yeniden belirlenir,

1. Adım 2 ve 3 merkezler sabit kalıncaya kadar tekrarlanır. K-means algoritması basitliği nedeniyle tercih edilen bir algoritmadır. Eğer elimizdeki örnekler $X=\{x^t\}^N_{t=1}$. K adet vektör v_j, $j=1,...,k$ ile ifade edildiğinde hata oranı b^t_i dir. Şekil 4.2'de algoritma detayları görülmektedir.



Şekil 4.2. K-means algoritması

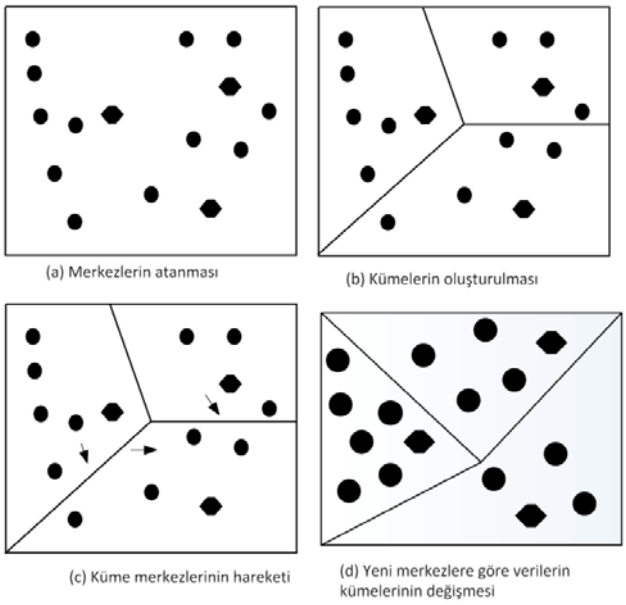
4.2.2. K-means++

K-means++ ilk kez 2007 yılında Arthur ve Vassilvitskii tarafından önerilmiştir [99]. K-means algoritması zayıf küme merkezlerinin iyileştirilmesini amaçlamaktaydı. Ancak bu algoritmanın bazı eksiklikleri vardı. Bunlardan biri yavaş olması ve istenildiği şekilde kümelerin oluşturulamamasıydı [100]. K-means++ algoritması dört adımda aşağıdaki şekilde açıklanabilir:

1. Birinci küme merkezi c_1 seçilir,
2. c_i merkezi $\frac{D(x)^2}{\sum_{x \epsilon X} D(x)^2}$ ile seçilir, xϵX,
3. İkinci adım tüm merkezler tanımlanıncaya kadar tekrarlanır,
4. Algoritmanın geri kalanı k-means adımlarıyla tamamlanır.

Burada $D(X)$ bir veri noktasının en yakın merkeze olan uzaklığıdır. K-means++

algoritması yerel minimuma yakınsar ki algoritmanın hesaplama karmaşıklığı *O(nkl)* ile ölçülür. Tekrar sayısı *l* ile simgelenir. *l* sayısı ilk oluşturulan merkeze bağlı olarak değişir. Toplam kümelenecek veri noktası *n* dir. *k* ise önceden tanımlanan küme sayısını ifade etmektedir. Karmaşıklık formülü $O(nkl)=nk\sum_{i=1}^{l} 1/i$ dir. Şekil 4.3'te k-means algoritmasına göre merkezlerin oluşturulması ve bu merkezlere göre verilerin kümelerinin değişimi görülmektedir. K-means++ algoritmasındaki tek fark Şekil 4.3'te başlangıçta oluşturulan üç küme merkezi yerine tek küme merkezinin keyfi atanması, atanan bu merkeze bağlı olarak diğer küme merkezlerinin oluşturulmasıdır.



Şekil 4.3. K-means algoritmasına göre merkez ve verilerin hareketi

4.3. Denetimli Öğrenme

Denetimli öğrenme, makine öğrenmesinin iki alt disiplininden biridir. Burada denetim kelimesinin geçmesi, öğrenme algoritmasının kontrol edildiği bir bilgiyi işaret etmektedir. Bu bilgi veri etiketidir. Etiketli veriler sayesinde denetimli öğrenme algoritmaları eğitilebilir. Eğer *x* verileri için *y* özellikleri ve *z* özellik vektörü verilmişse bunlara ek olarak *t* sınıf etiketi yer alır. Etiketleme bir veri noktası veya örneği için geçerli olup tek etiket tüm veriler için geçerli değildir. Tezin ele aldığı sınıflandırma probleminde deneysel veriler ikili etiketlenmiş verilerdir. Nitekim burada ikili

sınıflandırma problemi var demektir. Tablo 4.1'de denetimli öğrenmeye uygun bir veri seti görülmektedir. Bu veri setinde *y*:2, *z*:4 ve *t*:4'tür. Her örnek bir yazılım birimine ait iki özelliği ifade eder. Denetimli öğrenme algoritması etiketlenmiş verileri bir haritalama fonksiyonu *f(x)* oluşturabilmek için kullanır. Bu fonksiyona rastgele bir vektör sunulduğunda etiketini tahmin eder. Tahmin başarısı doğal olarak %100 beklenmez. Ancak temel hedef tahmin hatasını en aza indirebilmektir. Algoritmanın daha önce görülmemiş bir verinin etiketini doğru tahmin edebilmesi için eğitim için sunulan verilerin genelliği önemlidir. Tablo 4.1'deki verileri inceleyecek olursak program karmaşıklığı *V(G)* 10'dan büyük bir örnek vektörünü *f(x)* fonksiyonu hatalı olarak etiketleyecektir.

Tablo 4.1. Denetimli öğrenmeye uygun bir veri seti

Satır sayısı	V(G)	Hata?
10	3	0
50	7	0
100	15	1
150	18	1

4.3.1. Örnek-tabanlı öğrenme

Örnek-tabanlı öğrenme her deneme örneğinin düzenli kaydedilmesini gerektirir. Bu tarz öğrenmede deneme verileri her tekrarda değiştirilmek zorundadır. Bunun temel avantajı verilerin dinamik olarak eklenme, değiştirme ve silme işlemlerinin yapılabilmesidir [101]. İşlemler sırasındaki ana sorun büyük verilerde algoritmanın yavaş çalışmasıdır. Tüm örnek-tabanlı öğrenme algoritmalarında bazı bileşenler esasdır. Birincisi, benzerlik veya uzaklık fonksiyonları yardımıyla deneme veri setleri arasında veya deneme ile test veri setleri arasındaki benzerlikler bulunmaya çalışılır. İkincisi, belirli bir test verisi üzerinden sınıflandırma yapılır. Üçüncüsü, sınıflandırmanın başarısını izleyen bir birim içerir.

Kullanılan uzaklık fonksiyonu, verilerin içerdiği özellik tiplerine göre değişmektedir. Sayısal veriler için genellikle öklit uzaklık fonksiyonu kullanılır. Eğer y_i ve y_j örnek vektörleri z adet özellik içeriyorsa genel öklit eşitliği (Denklem 4.1)'deki gibi hesaplanır.

$$ED(y_i, y_j)=\sqrt{\sum_{d=1}^{z}(y_i^d - y_j^d)^2} \quad (4.1)$$

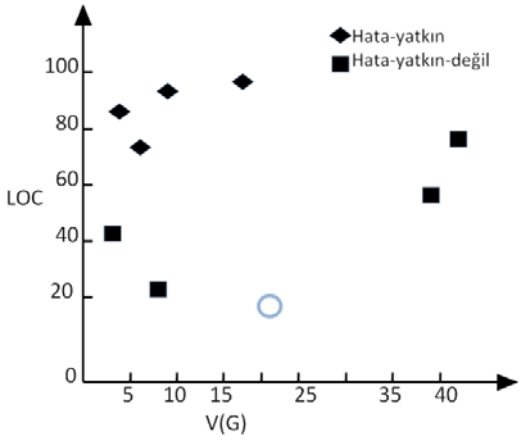
4.3.1.1. Ezberci öğrenme

Ezberci öğrenme eğitim verilerinin belli veritabanlarına kaydedilmesi esasına dayanır [102]. Dolayısıyla kaydedilen bilgilerin tekrar çağrılması bu tip öğrenmede oldukça etkilidir. Buradaki tek kısıt verilerin vektörel olarak depolanabilir olmasıdır. Verilerden bir çıkarım veya genelleme yapılmaz. Doğru etiketlenmiş verilerin özellikleri bilinebilir. Etiketleme hatalarının fazla olduğu verilerde bu tip öğrenme uygun değildir.

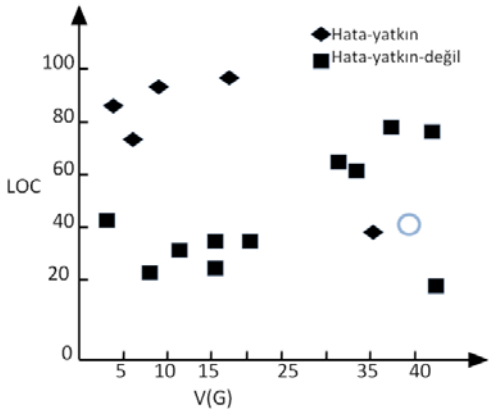
4.3.1.2. En yakın-komşuluk-tabanlı öğrenme

Ezberci öğrenmeden çok daha güçlü bir metot da en yakın-komşuluk-tabanlı öğrenmedir. Temel algoritması tek-komşuluk tabanlıdır. Burada test aşamasında eğitimde bulunan vektörlere en yakın vektörler bulunmaya çalışılır [103]. Şekil 4.4 ve Şekil 4.5'te en yakın komşuluk örnekleri sunulmuştur.

En yakın komşuluğun genel biçimi k-en yakın-komşuluk şeklindedir. k>=1 deneme örnekleri bulunur, test vektörleri ile denenir. k değerinin tek olması ikili sınıflandırma problemlerine uygun olduğu anlamına gelmektedir. Daha net çözümler elde edebilmek için k değerinin olabildiğince yüksek seçilmesi gerekir. Bu seçim test vektörünü çevreleyen deneme örneklerinden daha fazla bilgi edinebilmek içindir. k değeri verilere bağlı olarak seçilmelidir. Bu değerin daha kolay tahmin edilebilmesi için bir model optimizasyonu kullanılabilir. En yakın-komşuluk algoritmaları sezgisel ve kolay anlaşılabilir olmalarına rağmen gürültü ve ilgisiz özelliklere karşı hassatırlar. Dolayısıyla bu algoritmaları uygulamadan önce veri temizleme aşamasının tamamlanması gerekmektedir. Sınıf dağılımı eğitim veri setinde yoğunsa test işleminde elde edilecek öğrenme daha büyük olacaktır. Sınıf dağılımı bu noktada önemlidir [104].



Şekil 4.4. En yakın-komşuluk örneği. Bu örnekte daire şeklindeki örnek en yakın-komşuluğa bakılarak hata-yatkın-değil şeklinde etiketlenecektir



Şekil 4.5. Hatalı en yakın-komşuluk örneği. Bu örnekte daire şeklindeki örnek hata-yatkın şeklinde etiketlenecektir, bu durum hatalı bir yakın komşudan kaynaklanmaktadır

4.3.1.3. Sorgu-tabanlı öğrenme

Sınıflandırıcıya bazı sorular yöneltilerek öğrenme işlemi gerçekleştirilmeye çalışılır. Buradan elde edilen cevaplarla bazı kehanetler ortaya atılır. Bu işleme karmaşık hesaplamalı simülatörler, pahalı bir deneyle veya insan bir uzman ile dahil edilir. Sorguları sistematik bir eğitim veri seti üzerinde görüntülersek interaktif öğrenme ortaya çıkar [105].

Eğer eğitim örnekleri *[x,o(x)]* olarak tanımlanırsa ki *x* giriş vektörü ve *o(x)* hedef çıkış vektörüdür. Sorgulara bağlı bir kehanet ile eğitim örnekleri modellenebilir [106]. *f* bilgisi verilirse *o(f)* sorgu yardımıyla bulunabilir. Sorgulanan örnekler de *[f,o(f)]* ile

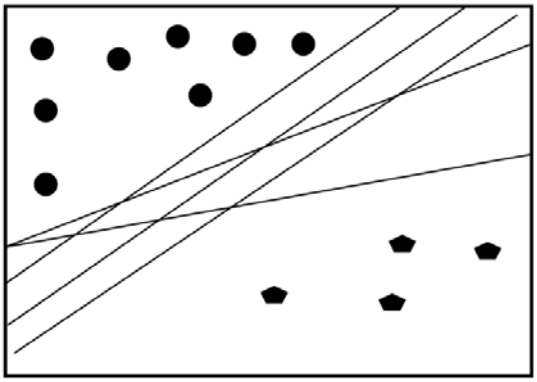
ifade dilir. Tablo 4.2'de sorgu tablosu örneği görülmektedir.

Tablo 4.2. Sorgu tablosu örneği

Sorgu ismi	**Sorgu**	**Cevapla:Evet**	**Cevapla:Hayır**
Aralık tahmini	*3<o(x)<10?*	"Evet"	"Hayır"
Eşitlik	*o(x)=o(f)?*	"Evet"	"Hayır"
Alt veri	*o(x)∩o(f)?*	"Evet"	"Hayır"

4.4. Lineer Ayrıştırıcılar

Lineer ayrıştırıcılar makine öğrenmesinin merkezinde yer alan çalışma alanlarından biri olup, hem istatistiksel hem de hesaplamalı öğrenme teorilerinde kullanılmaktadır [107]. Lineer ayrıştırma teknikleri sayısal özelliklerin lineer kombinasyonlarını kullanarak bu özellikleri sınıflara böler. Bu sırada birden fazla seçenek oluşabilir. Dolayısıyla ayrıştırma işlemi bir optimizasyon gerektirir [108]. Çözüm seçenekleri sayısız olabileceği için lineer ayrıştırıcılar için geliştirilen algoritmalar deterministik değildir [109]. Bunu Şekil 4.6 ile açıklayabiliriz. Burada iki veri grubunu ayrıştırmak için birçok seçenek mevcuttur.



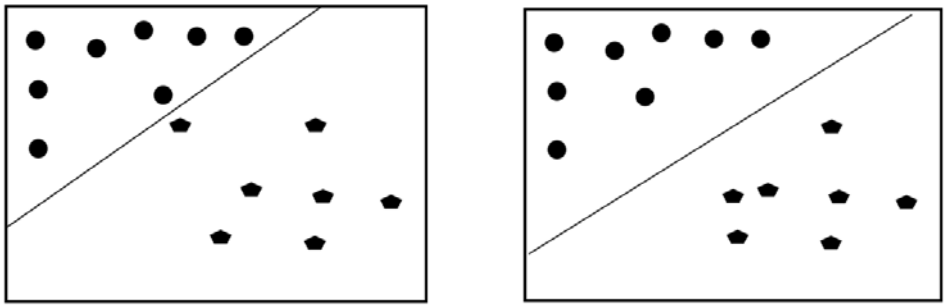
Şekil 4.6. Lineer ayrıştırıcı seçenekleri

İdeal ayrıştırıcının görülmeyen veriler üzerinde de yüksek performans göstermesi beklenir. En iyi performansının elde edilip edilmediğini iki sınıf arasındaki mesafeye bakarak ölçebiliriz. Lineer ayrıştırma sonrası iki sınıf arasındaki mesafe ne kadar fazlaysa o kadar iyi bir performans elde edilmiş demektir.

Eşitlikler yardımıyla ayrıştırıcılar ifade edilmek istenirse eğer bir *X* örneği bir *K* sınıfına aitse ve bu test edilecekse, sınıflandırma özelliklerini *Y* ve *Z* ile açıklayalım. Tahmin edici özelliklerin tamamının sayısal olduğu varsayılır. *X* örneği vektörel olarak *<X1..Xn>, Xi* sayısaldır. Eğer vektörel değerlerin ağırlıklandırma yardımıyla ifadesi mümkünse $W_1X_1+ W_2X_2+...+WnXn>Y$ koşulu sağlanıyorsa *X.K=Y*, aksi durumda *X.K=Z*.

Maliyet açısından değerlendirecek olursak çok eğitim hatası ve geniş mesafede maliyet az, az eğitim hatası ve küçük mesafede maliyet fazladır.

Lineer ayrıştırıcıların avantajlarından biri aynı kovaryanslı normal dağılımlar için en uygun yöntem olmasıdır. Diğer taraftan bu ayrıştırıcılar lineer ayrışabilir problemlerle sınırlıdırlar. Özellik sayısının fazla olduğu problemler örnek verilebilir. Hem özellik sayısının fazla olması, hem de veri sayısının fazla olması bu tip problemlerde hesaplama verimini arttırmaktadır. Şekil 4.7'de mesafe tiplerine göre ayrıştırıcı örnekleri gösterilmiştir.

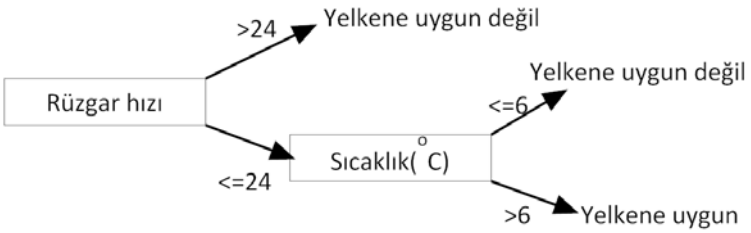


Şekil 4.7. Lineer ayrıştırıcı örnekleri, a) Küçük mesafeli tüm eğitim örnekleri doğru sınıflandırılmış b) Büyük mesafeli eğitim hatalarına izin verilmiş

4.5. Ağaç-tabanlı Öğrenme

Ağaç-tabanlı öğrenme karar ağaçları olarak da adlandırılmakta olup tahmin işleminin yapılacağı eğitim verilerinden yapısal bir sınıflandırma oluşturulmaya çalışılır. Verilere yukarıdan aşağıya ağaç oluşturacak şekilde bir yöntem uygulanır [110]. Bu yukarıdan aşağıya yaklaşım böl ve yönet olarak da adlandırılmaktadır [111]. Oluşturulan ağaçlar

öz yinelemeli olarak çözümlenmektedir. Kök ve zirve düğümlere karar vermek için çeşitli algoritmalar geliştirilmiştir [112]. Burada temel mantık ağaçtaki bir önceki düğüme bakarak buna bağlı düğümleri oluşturmaktır. Eğer eski düğümler *h* ile ifade edilirse *P(f|h)* olasılığında *f* gelecek düğümü simgeler. Geleneksel yelkenci problemi karar ağacı ile çözülerek örneklenir. Burada rüzgar hızı ve sıcaklık değerlerine bakılarak yelkene uygunluk tespit edilmeye çalışılır. Şekil 4.8'de bu değerler karar ağacında sorgulanarak yelken uygunluğu test edilmeye çalışılır. Ağaca bakılarak yelken için rüzgarın 24(mph)'dan küçük ve sıcaklığın 6 dereceden büyük olması gerekmektedir. Bu test işleminin farklı durumlar için tekrarlanması gerekmektedir.



Şekil 4.8. Karar ağacı örneği [112]

4.5.1. C4.5 algoritması

Quinlan tarafından 1993 yılında geliştirilen bir algoritmadır [113]. C4.5 farklı kararları bulabilmek için özellik aralığının minimum kombinasyonu dışındaki herşeyi ihmal eder. C4.5 böl-yönet yaklaşımını temel alarak karar ağacını oluşturur [114]. Eğer *F* durumlarından bir ağaç oluşturulacaksa:

1. *F* bir durumra kriterini içeriyorsa, *F* içindeki en sık kullanılan sınıflar ile *D* ağaçları yaprak bağlantılı demektir.
2. Bazı *T* test durumları $T_1,...,T_n$ *D* alt durumlarını $D_1,...,D_n$ bölmek için kullanılır, D_i durumları T_i sonuçlarını üretir.

Öz yinelemeli bölümleme stratejisi eğitim verileri ile tutarlı olabilir. Pratikte veriler gürültü içerdiği için özellik değerleri yanlış kaydedilmiş ve sınıflandırma hatalı olabilir. İlk ağaç birçok sistemde budanır, tahmin kesinliğine az katkı yapan ağaçlar yapraklar ile değiştirilir.

4.5.2. CART algoritması

Basit veri madenciliği problemlerinin çözümü için geliştirilen bir diğer öğrenme algoriması CART'dır [115]. C4.5 gibi bu algoritma da ağaç tabanlı bir algoritmadır. C4.5 farklı algoritmaların (ör: rastgele orman) geliştirilmesi için ilham kaynağı olmuştur. Ağaç yapısında kullanılan gerileme yöntemi açısından C4.5 ve CART benzerdir. Ancak CART algoritmasında hedef sınıf sayısaldır. CART ağacındaki yapraklar dolayısıyla sayısal değerler içerir. Buradaki çeşitlilik özellikle sayısal hedefler için standart sapma ile kolaylıkla bulunabilir.

Sayısal hedefler sınıf dağılımında $x_1,x_2,x_3,...,x_n$ şeklinde ifade edildiğinde n ölçüm için t tüm x değişkenlerinin toplamı, t_2 de tüm x değişkenlerinin kareleri toplamı ise standart sapma (Denklem 4.2)'deki gibi bulunur.

$$\text{ss}=\sqrt{\frac{t_2-(t^2/n)}{n-1}} \tag{4.2}$$

C4.5 ve CART birbirine çok benzeyen iki algoritmadırlar. Algoritmada beklenen çeşitlilik hesabındaki tek fark C4.5 algoritmasının entropi, CART algoritmasının ise standart sapmayı kullanmasıdır. (Denklem 4.3)'te bu fark formül ile gösterilmiştir. C4.5 algoritmasının Java programlama dili ile yazılmış şekli J48 olarak adlandırılır.

$$ed = \sum_i \frac{n_i}{n} * (e_i \ veya \ s_i) \tag{4.3}$$

4.5.3. Rastgele Orman algoritması

Thomas Bayes'in yöntemi temel alınarak geliştirilmiş bir sınıflandırıcıdır [116]. Ağaç tahmin edicilerin bir kombinasyonu olan Rastgele Orman algoritmasında her ağaç rastgele alınmış bir vektöre bağlıdır. Bu sınıflandırma biçiminde giriş değerleri ve özellikler rastgele alınmaktadır [117]. Alt yapısı Bayes teoremine dayanan bir mekanizma içerir. Büyük veri setleri için sınıflandırmadaki hata oranı küçüktür.

Geleneksel CART ve C4.5 algoritmaları büyük veri problemleri için yetersizdir [118]. Bu algoritmalar verinin tamamının ana belleğe yüklenmesi ve tek iş parçacığı ile çalıştırılması esasına dayanır. Tüm özellik sayısından daha az miktarda özellik seçilir. Seçilen özellik sayısı kadar özellik içeren ağaç yapıları oluşturulur.

Rastgele Orman algoritması kullanıcının anlayıp yorumlayabileceği tek model oluşturmaz. Tüm ağaçlarda kullanılan özellikler sorgulanır. Diğer taraftan bu algoritmayı içeren yazılım araçları verinin tümünün kullanılmadan önce belleğe yüklenmesini ister. Bütün bunlara rağmen Rastegele Orman algoritması diğer öğrenme algoritmaları ile karşılaştırıldığında oldukça hızlıdır. Örneğin Breiman [117], 50000 durumlu ve 100 değişkenli 100 ağacı 11 dakika içinde 800Mhz hızındaki işlemci ile oluşturabilmiştir. Günümüzün bilgisayarları ile bu deney tekrarlanırsa bu zamanın daha da kısalacağı açıktır. Farklı işlemciler üzerinde bu algoritma kullanılarak ağaçlar oluşturulabildiği için bulut bilişim sistemlerine de uygundur.

Torbalama (bagging) olarak adlandırılan bir topluluk metodu da Rastgele Orman algoritmalarının konusu altında yer alır [119]. Boosting gibi metotlarda geçmiş sınıflandırma performanslarına bakılarak her düğüme bir ağırlık verilir. Her eğitim verisi yeniden örneklenerek bootstrap yöntemi geliştirilmiştir. Her tekrarda eğitim verileri değiştirilir ve rastgele yeni özellikler getirilir. Bu yaklaşım çoklu ağaç yapısı ile Rastgele Orman algoritmasını birleştirmiştir [120].

4.6. Bayes Sınıflandırıcıları

Lineer sınıflandırma tekniklerinden biri olan Bayes sınıflandırıcısı ilk olarak Thomas Bayes tarafından geliştirilmiştir [121]. Bayes sınıflandırıcısı olasılık teorisi kullanılarak geliştirilen bir algoritma olduğu için olası durumları tanımlayan bir metot içerir. Teorem (Denklem 4.4) eşitliği ile ifade edilir. Teoreme göre S_{t1} eğitim veri seti ve S_{t2} test veri seti olarak ifade edilirse $S_{t1} \cap S_{t2} = \emptyset$.

$$P(A|B) = \frac{P(B|A).P(A)}{P(B)} \tag{4.4}$$

En popüler Bayes algoritmalarından biri Naive Bayes algoritmasıdır. İsminde naive kelimesinin kullanılmasının nedeni sınıflandırma için kullanılacak verilerin tümünün istatistiksel olarak birbirinden farklı olduğunu varsaymasıdır. Zaman içerisinde örnek-tabanlı algoritmalara benzer olarak Naive Bayes algoritması da kendini güncellemiştir. Naive Bayes eğitim veri setinden bağımsız çalışır. Hesaplama karmaşıklığının az olması ve büyük verilerle iyi çalışması bu algoritmanın tercih edilme nedenlerindendir. Text sınıflandırma temel kullanım alanlarındandır. Bunun yanında spam fitreleme ve medikal araştırma gibi farklı alanlarda da kullanılmaktadır [122], [123]. Eğitim sırasında, her özelliğin istatistiksel ağırlığı sınıflandırmaya dahil edilir. Bu işlemin avantajı her özelliğin ihtimalinin açıkça kaydedilebilmesidir. Modelin çalışma yapısını yorumlamayı kolaylaştırdığı için Naive Bayes Beyaz Kutu (White-Box Testing) yaklaşımı kapsamında değerlendirilebilir.

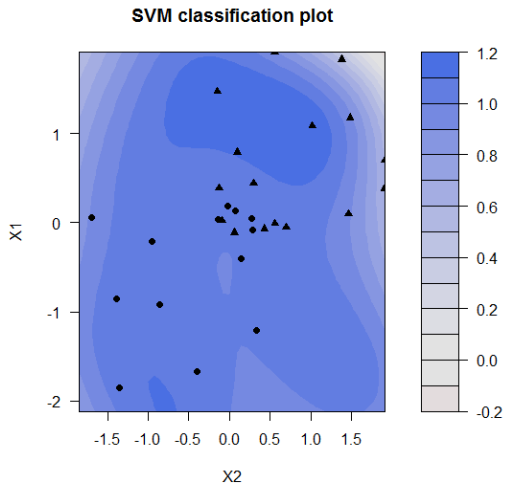
Her test vektörünün hangi sınıfa ait olduğunu Naive Bayes sınıflandırıcısı belirler. En yüksek olasılığa sahip sınıf tahmin edilen sınıftır. Hızlı ve kolay geliştirilebilir olması niave Bayes'in avantajlarındandır.

4.7. Karar Destek Makinaları

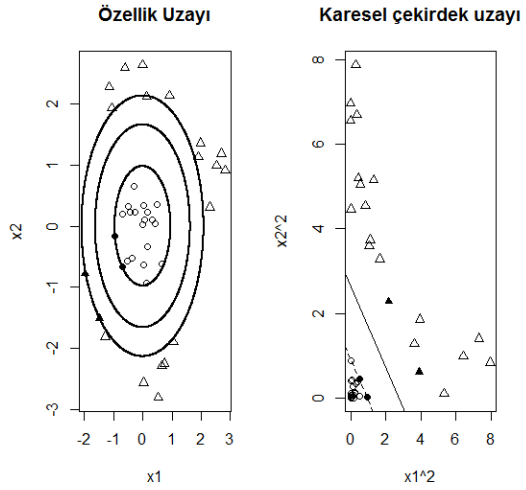
Karar destek makinaları (SVM) 1992 yılında Boser ve arkadaşları tarafından ortaya atılmıştır [124]. SVM, hiperdüzlemsel uzayda sınıfların bölünmesini öğrenen bir algoritmadır. Oldukça karmaşık veri setleri ile işlem yapabilmektedir. SVM algoritmaları sınıfları en iyi ayıran düzlemleri araştırırlar. En büyük mesafeli SVM, hiperdüzlem ile destek vektörünün mesafesini maksimum yapmaya çalışır. Başlangıçta lineer problemlere uygun gibi görünen SVM, daha sonra lineer olmayan sınıflandırma için başarı ile uygulanmıştır. Bir çekirdek fonksiyon yardımıyla çok yüksek-boyutlu uzayda veriler lineer olarak ayrıştırılabilmektedir [125].

Çekirdek fonksiyonları polinom, gaussian veya sigmodial şekilde olabilir [126]. Her birinin farklı karakteristiği vardır ve farklı bir probleme uygundur. Gaussian rbf modelini içeren SVM Şekil 4.9'da görülmektedir. Lineer olmayan uzaylara ait bölümleme Şekil 4.10'da verilmiştir. SVM tahmin kesinliği yüksektir ve aşırı-uygunluk

durumu azdır, gürültüye karşı duyarlıdır. Bununla berarber hesaplama maliyeti yüksek ve yavaş bir algoritmadır.



Şekil 4.9. Eş yükselti eğrilerini içeren karar değerlerinin SVM ile ikili sınıflandırılması

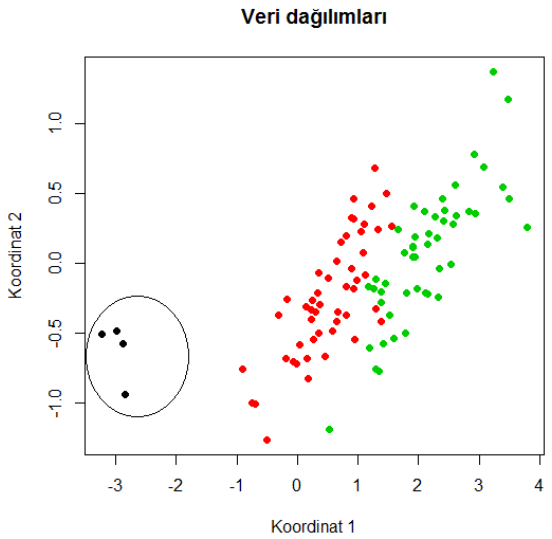


Şekil 4.10. Lineer olmayan alanların karesel çekirdek uzayında bölümlenmesi

4.8. Dengesiz Sınıf Dağılımı Problemi

Sınıflandırıcıların kesinliğine etki eden unsurlardan biri dengesiz sınıf dağılım problemidir [127]. Eğer etiketli veri oranı seyrekse sınıflandırıcı bu bölümü görmezden gelerek çoğunluğun bulunduğu sınıfı tahmin etmeye çalışacaktır. Bunun başlıca sebeplerinden biri sınıflandırıcıların yapıları gereğidir. Bir sınıflandırıcı eğitim verileri

için en çok hangi sınıf mevcutsa o sınıfı tahmin etmeye uygun olarak yapılandırılır. Örneğin elimizdeki veri setinde her 200 veriden sadece biri pozitif örnek içeriyorsa, sınıflandırıcı negatif örnekleri tahmin edecektir. Bunu aşabilmek için kesinlik performans parametresi yerine F-ölçüt kullanılabilir, algoritmanın parametreleri değitirilebilir veya eğitim verileri üzerinde örnekleme yöntemleri uygulanabilir. Örnekleme uygulanırken ya çoğunluk sınıftan veriler silinebilir ya da az gruba yeni veriler eklenerek çoğaltılmaya çalışılır.



Şekil 4.11. Dengesiz veri dağılımı

Şekil 4.11'de dengesiz veri dağılımını içeren bir veri grubu gösterilmektedir. Burada çember içine alınan grup ile diğer grup karşılaştırıldığında çember içindeki grup diğer veri sınıfına göre oldukça azdır. Dolayısıyla sınıflandırıcı siyah renkle simgelenen noktalar yerine diğer iki sınıfı tahmin etmeye çalışacaktır.

4.8.1. SMOTE algoritması

Az sayıdaki eğitim verilerinin güvenilir sınıflandırma sonucu elde edebilmek için çoğaltılmasına yönelik yöntemlerden biri SMOTE algoritmasıdır [128]. El yazısı karakterlerin tanınmasına yönelik geliştirilen bir yöntemden ilham alınarak geliştirilen algoritma gerçeğe yakın eğitim verileri üretir. Gereken eğitim veri seti büyüklüğüne

bağlı olarak *k*-komşuluk adedi seçilir. SMOTE algoritma adımları Şekil 4.12'de görülmektedir.

```
SMOTE(T,N,k)
Giriş: Az sayıdaki sınıfa ait örnekler T, Smote oranı N, komşuluk
k
Çıkış: (N/100)*T
if N<100
      then T grubunu rastgele seç,
           T=(N/100)*T
           N=100
end if
Ornek[][] : ilk küçük sınıf grubu
uretilen: uretilen ornekler
sanal[][]: sanal örnekler
for i:1 to T
       tüm i lemanları için en yakın k hesapla ndizi'ye ekle
       Türet(N,i,ndizi)
endfor
Türet(N,i,ndizi){
while N!=0
         for özellik=1:özellikSayisi
              Hesapla:fark=Ornek[ndizi[adet][ozellikSayisi]-Onek[i]
             sanal[uretilen][ozellik]=ornek[i][ozellik]+fark
          endfor
         uretilen++
          N=N-1
endwhile
return
}
```

Şekil 4.12. SMOTE algoritma adımları

4.8.2. Virtual algoritması

Hata veri setlerinde dengesiz veri dağılımına çözüm olabilecek bir diğer algoritma Virtual'dır. Bu algoritma veri örnekleme ve aktif öğrenme yöntemlerini içerir [129]. Tüm örnekleme işlemleri eğitim sürecinden önce tamamlanır. SVM tabanlı geliştirilen algoritma g-mean performans parametresi açısından SMOTE algoritmasından üstün bir algoritmadır [16]. Literatürde hata veri setleri üzerinde uygulanmamış bir algoritma olarak görülmektedir.

Virtual algoritması öncelikle rastgele bir örnek havuzu oluşturur. Oluşturulan bu havuzdan rastgele seçilen örnek *hyperplane* fonksiyonu ile diğer örneklerle olan mesafeye bakılır. En küçük mesafeli örnek kaydedilir. Kaydedilen bu örnek daha sonra SV olarak işleme sokularak k-nn belirlenir. En yakın komşuluk ile belirlenen örnek ve SV olarak alınan örnek 0-1 arası türetilen bir değer yardımıyla oluşturulacak yeni örnek için kullanılır. Bu işlemler dizisi istenilen düzeyde örnek adedine ulaşılıncaya kadar tekrarlanır. Bu algoritmanın avantajlarından biri eğitim zamanının kısa olmasıdır. Ayrıca eğitimin erken safhalarında destek vektörlerinin işlem yapması üretilen sanal örneklerin optimizasyonuna yardımcı olmaktadır.

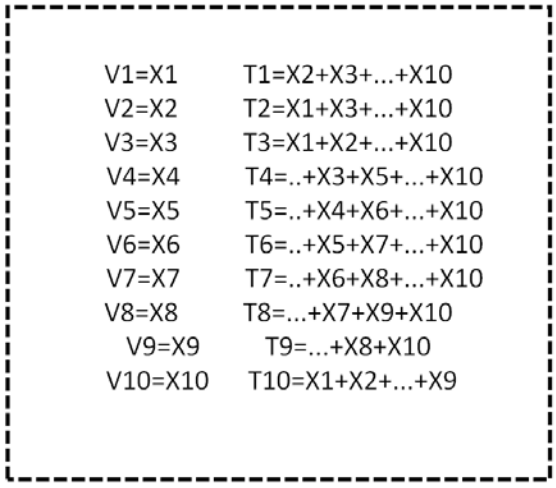
4.9. Sınıflandırıcı performansının ölçümü

4.9.1 Çapraz onaylama (cross-validation)

Çapraz doğrulama sınıflandırmanın güvenilirliğini arrtırmak, özellikle aşırı-uygunluk durumlarını çözmek için kullanılan bir yöntemdir [130]. Bir veri seti grubu *X*'den öncelikle eğitim ve test grubu alınır. Eğer veri seti *X* yeterince büyükse *K* grubuna bölünebilir. Bu grubun bir bölümü eğitim diğer bölümü ise test için kullanılır. *K* genellikle 10 veya 30 olabilir. Bu ayrım oranları mümkün olmadığında aynı veriler tekrar kullanılır. Bu durum çapraz onaylama olarak adlandırılır.

Çapraz onaylamada hata oranını düşürebilmek için eğitim ve test veri grupları olabildiğince büyük seçilmeye çalışılır. Böylece hata oranı azalır.

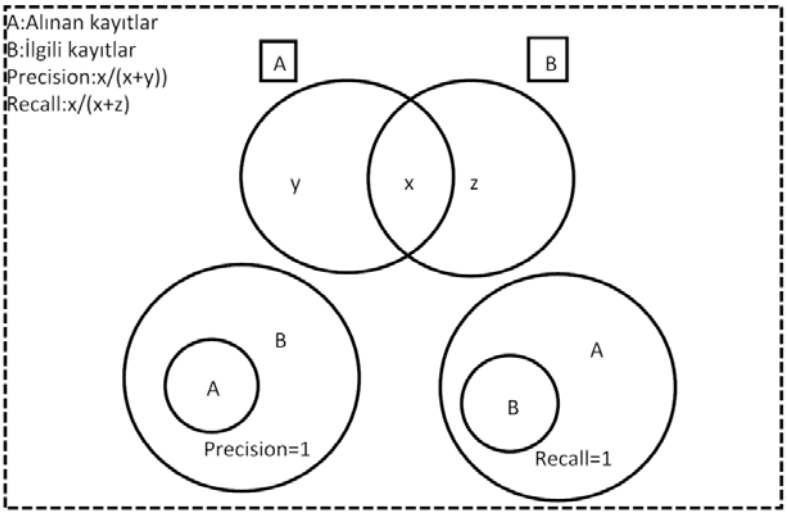
K-kat çapraz onaylamada veri seti *K* adet gruba bölünür $X_i, i=1,...,K$. Bu veri gruplarından biri test için, *K*-1 adedi ise eğitim için ayrılır. İşlemin tekrar adedi K'yı belirler. Örneğin 10 kez tekrarlandığında 10-kat çapraz onaylama (10-fold cross-validation) olarak adlandırılır. Şeki 4.13'deki adımlara bakacak olursak test için ayrılan grup *V* eğitim grubunda yer almaz ve her eğitim grubunun *T,* 8 adet grubu aynıdır.



Şekil 4.13. Çapraz onaylama adımları

4.9.2. Kesinlik-Geri çağırma performans parametreleri

Arama etkiliğini ölçmek için kullanılan performans parametrelerinden arama etkiliğini açısından kesinlik ve Geri çağırma sıklıkla kullanılmaktadır [59]. Bu performans parametrelerin alınan verilerin ilgililik düzeyini göstermektedir. Bir veri grubundan tüm ilgili verileri alabilmek ve değersiz verilerin tamamını eleyebilmek mümkün değildir. Ancak bu işlemlerin başarı oranı bir şekilde ölçülmelidir.



Şekil 4.14. Kesinlik-Geri çağırma hesaplamalarının şema ile gösterimi

Şekil 4.14'de kesinlik ve geri çağırma parametrelerine ait hesaplama yöntemleri şema ile gösterilmiştir. Burada tüm alınan kayıtlar A, tüm ilişkili kayıtlar B ile simgelendiğinde, tüm ilişkili alınan kayıt x, alınan fakat ilişkili olmayan kayıt y, alınmayan fakat ilişkili olan kayıtlar ise z ile gösterilir. Kesinlik ve Geri çağırma parametrelerinin 1 değerine sahip olduğu durumlar da ayrıca küme biçiminde gösterilmiştir. Kesinlik ve Geri çağırma parametreleri birbiriyle ters orantılı iki parametredir. Dolayısıyla asıl hedef iki parametrenin de en uygun noktasını yakalayabilmektir. Bunun için alınan verilerin belirli bir noktada durdurulması ve veri alış yönteminin iyi belirlenmesi gerekmektedir.

Geri çağırma parametresini hesaplayabilmek zordur, çünkü veri tabanında ne kadar ilişkili veri olduğunu tespit etmek de zordur. İkili verilere ait bir havuz yardımıyla bu

oran tahmin edilmeye çalışılır. Bu noktada araştırmacı kendine has arama yöntemleri geliştirmek zorundadır.

4.9.3. Karışıklık matrisi

Hata tahmini ikili sınıflandırmaya dayalı gerçekleştirildiği için eğer performans değerlendirilmesi yapılacaksa öncelikle karışıklık matrisi (confusion matrix) oluşturulması gerekir. Tahmin sırasında yazılım modülleri hakkında hata yatkın(fp) veya hata yatkın değil(nfp) şeklinde tanımlamalar yapılır. Eğer bir modül hata içermeyip doğru hüküm verildiyse TN, hata içerdiği halde yanlış hüküm verildiyse FN, hata içermeyip yanlış hüküm verildiyse FP, hem hata içerip hem de doğru hüküm verildiyse TP olarak etiketlenir. Bu tanımlamaların ardından Tablo 4.3 oluşturulur.

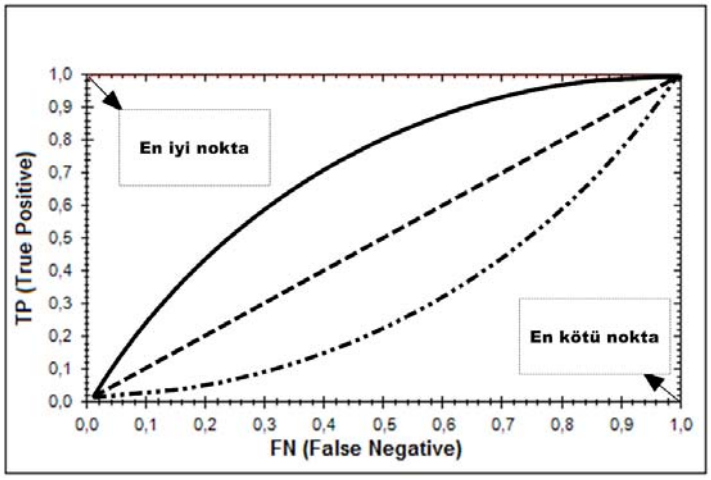
Tablo 4.3. Karışıklık matrisi

		Tahmin	
		nfp	fp
Gerçek	nfp	TN	FP
	fp	FN	TP

Hata tahmininde performans analizi için kullanılan bir diğer terim İşlem karakteristik Eğrisi (Receiver Operating Characteristics ROC) dir [131]. İlk olarak sinyal belirleme teorisinde kullanılan ROC daha sonra birçok farklı alanda özellikle sağlık alanında yapılan karar-tabanlı çalışmalarda kullanılmıştır. Makine öğrenmesi ve veri madenciliğinde model değerlendirmesi için tercih edilen bir parametre olmuştur [132], [133].

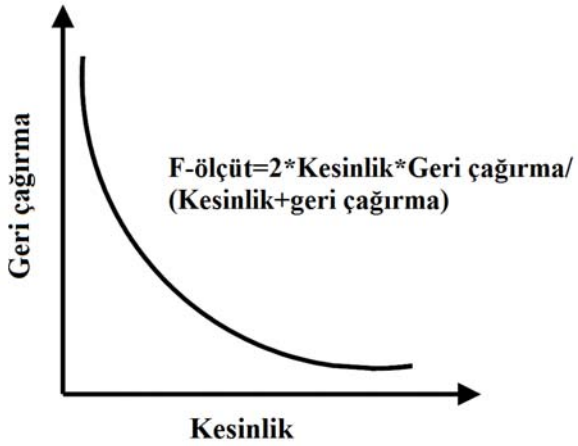
Şekil 4.15'te ikili sınıflandırma başarısını ölçen ROC eğrisi görülmektedir. Y ekseninde ifade edilen değer TP/TP+FN formulü ile bulunur. Aynı şekilde x ekseninde ifade edilen değer FP/FP+TN formulü ile bulunur. Bir önceki bölümde tanımladığımız arama etkililiğini ölçerken kullanılan Geri çağırma ve FPR değerlerinin ROC eğrisinde de kullanıldığı anlaşılmaktadır. Eğrinin altında kalan alan Area Under Curve(AUC) ile ifade edilir. (1,0) noktası en iyi nokta olup hiç yanlış tahminde bulunulmadığı anlamına gelir. (0,1) noktası da en kötü nokta olup hiç doğru tahminde bulunulamadığı anlaşılır.

Köşegenin üstünde kalan eğri genellikle arzulanan sonuçtur. Köşegenin altında kalan eğri ise başarının yeterli düzeyde olmadığını gösterir.



Şekil 4.15. ROC eğrisi

Kesinlik ve Geri çağırma değerleri birbiriyle ters orantılı olup üretilen genel sonuç F-ölçüt olarak adlandırılmakta ve bu iki değerin harmonik ortalaması hesaplanarak Şekil 4.16'daki gibi elde edilmektedir. F-ölçüt değeri 1'e yaklaştıkça daha iyi performans elde edilmiş demektir.



Şekil 4.16. F-ölçüt

Makine öğrenmesi performans değerlendirme algoritmalarından biri de g-ortalama dır. Hassasiyet ve özgüllük değerlerinin geometrik ortalamasıdır. Bu performans parametresi özellikle dengesiz veri setlerinde kullanılmaktadır. Ayrıca öğrenme algoritmalarının başarım karşılaştırmasında da kullanılır. Hata tahmininde kullanılan performas parametresi formüllerinin özeti Tablo 4.4'te görülmektedir. Tablo

incelendiğinde ilk göze çarpan bazı performas parametresi formülleri aynıdır. Bunlar TPR-Hassasiyet- Geri çağırma ile TNR-Özgüllük parametreleridir. Literatürdeki çalışmalarda formüllerden bazıları seçilerek performans değerlendirmesi yapılmıştır. Hangi performans parametresinin seçileceği kullanılan yönteme bağlı olarak değişmekle birlikte seçim yöntemiyle ilgili herhangi bir standart bulunmamaktadır. Ancak parametreler arasındaki farkı tespit edebilmek için ANOVA gibi istatistiksel yöntemler kullanılabilir. Sonraki bölümde ANOVA yöntemi detaylandırılmıştır.

Tablo 4.4. Performas parametre formülleri

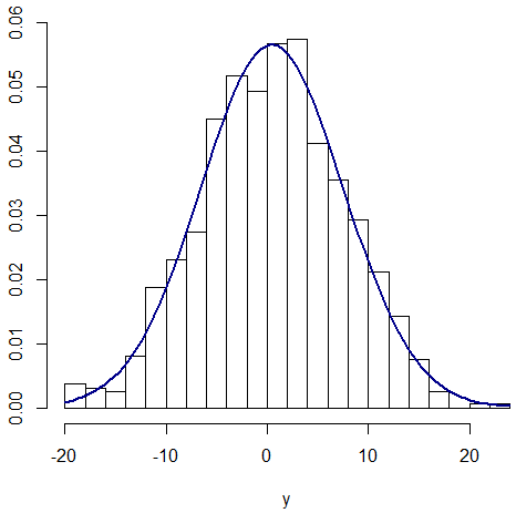
İsim	Formül
TPR Hassasiyet(Sensitivity) Recall	TP/(TP+FN)
FPR	FP/(FP+TN)
Kesinlik	TP/(TP+FP)
TNR Özgüllük(Specificity)	TN/(TN+FP)
F-ölçüt	$\frac{2 * Recall * Precision}{Recall + Precision}$
Doğruluk(Accuracy)	TN+TP/(TN+FN+FP+TP)
Geometrik ortalama(G-mean)	$\sqrt{Hassasiyet * Özgüllük}$

4.10. İstatistiksel Metotlar

4.10.1. ANOVA

ANOVA sayısal veri grupları arasında veya grup içinde farklı değerleri tespit edebilmek için kullanılan bir istatistiksel analiz yöntemidir [134]. ANOVA analizini gerçekleştirebilmek için verilerin belirli özelliklere sahip olması gerekir. Bunlardan ilki verilerin her birinin birbirinden bağımsız olmasıdır. Veriler normal diğer adıyla Gaussian dağılımına uymalıdır. Şekil 4.17'de normal dağılıma uygun verilere ait grafik verilmiştir. Örneklerin varyansları farklı olmalıdır. ANOVA'nın kesinlik içermediği durumlar da mevcuttur. Eğer verilerde kayda değer bir fark bulunamazsa veriler aynıdır hükmüne varılamaz. Ayrıca ANOVA veri gruplarının farklı olup olmadığını bulurken hangi veri grubunun farklı olduğunu tespit edemez. Bunun için farklı testler uygulanmalıdır (ör: çoklu karşılaştırma testleri).

Burada *j* grup indeksini gösterir ve grup içindeki bir bölümün analiz değeri Y_{ij} ile temsil edilir. Bölüm *i* le gösterilir. $\bar{Y}.j.$ analizde herhangi bit gruba ait ortalama değerini temsil etmektedir. Burada "." kullanılmasının nedeni o gruba ait tüm bölümlerde analizin yapılmasıdır.



Şekil 4.17. Normal dağılım grafiği

Tüm bölümlerde ve gruplarda yapılan bir analiz olduğu için tüm örneğin ortalaması $\bar{Y}..$ ile gösterilir. "." Analizin tüm bölümlerde veya gruplarda yapıldığını göstermektedir. Gruplar arasındaki kareler farkı (Denklem 4.5) ile hesaplanır. Bu eşitlikteki gözlem sayısı *n* dir.

$$'S_A = n\sum(\overline{Y.J} - Y..)^2 \tag{4.5}$$

Grupların kareler toplamı (Denklem 4.6) ile elde edilir:

$$SS_T = SS_A + SS_{S/A} \tag{4.6}$$

ANOVA hesaplamalarındaki kareler toplamı işlemleri belirli oranda hata içerir. Bu hatalar (Denklem 4.7)'deki gibi tanımlanır:

$$SS_{S/A} = \sum.\sum(\overline{Y\,ij} - Y.J)^2 \tag{4.7}$$

$$df_A = a - 1;\ df_{S/A} = a(n-1) = N - a \tag{4.8}$$

Her kareler toplamının (Denklem 4.8)'deki gibi bir özgürlük derecesi vardır ve bu derece tektir. Grup sayısı *a* ile simgelendiğinde *n* gruptaki gözlem sayısıdır. Eşitlikteki toplam gözlem sayısı N'dir. Gözlem sayısı her grupta hesaplanır. F-skor değeri gruplar arası varyansın grup içi varyansa bölümü ile elde edilir. F-skor formülü (Denklem 4.9) görülmektedir.

$$F = \frac{MSA}{MS\frac{S}{A}} \tag{4.9}$$

ANOVA tablosu Tablo 4.5'teki gibi oluşturulur. Tabloda toplam sekiz parametre bulunmaktadır.

Tablo 4.5. ANOVA

SA	fA	1S$_A$
S$_{S/A}$	f$_{S/A}$	1S$_{S/A}$
S$_T$		

Elimizde üç grup yazılıma ait beş farklı özellikle ilgili değerler olduğunu varsayarak ANOVA tablosu oluşturalım. Yazılım proje isimleri eclipsePde, equinox ve lucene'dir. Bu üç gruba ait özellik değeleri ile hesaplama tabloları ANOVA analizine uygun olarak Tablo 4.6'daki gibi hesaplanır ve Tablo 4.7'deki gibi özetlenir.

Tablo 4.6. ANOVA parametre hesapları ve veriler

eclipsePde	$(Y_{ij}-\bar{Y}_{.1})^2$	$(Y_{ij}-\bar{Y}_{..})^2$	**equinox**	$(Y_{ij}-\bar{Y}_{.2})^2$	$(Y_{ij}-\bar{Y}_{..})^2$	**lucene**	$(Y_{ij}-\bar{Y}_{.3})^2$	$(Y_{ij}-\bar{Y}_{..})^2$
4	4	9	9	0	4	6	0	1
4	4	9	8	1	1	6	0	1
6	0	1	10	1	9	6	0	1
8	4	1	8	1	1	7	1	0
8	4	1	10	1	9	5	1	4
$\bar{Y}_{.1} = 6$	$\sum(Y_{ij} - \bar{Y}_{.1})^2$=16		$\bar{Y}_{.2} = 9$	$\sum(Y_{ij} - \bar{Y}_{.2})^2$=4		$\bar{Y}_{.3} = 6$	$\sum(Y_{ij} - \bar{Y}_{.3})^2$=2	

Tablo 4.7. ANOVA özet tablo

$SS_A = n\sum(Y_{ij} - \bar{Y}_{..})^2 = 5[(6-7)^2+(9-7)^2+(6-7)^2] = 30$	$df = a-1 = 3-1 = 2$	$MS_A = \frac{SSA}{dfA} = \frac{30}{2} = 15$	$F = \frac{MSA}{MS_{S/A}} = 15/1.83 = 8.182$
$SS_{S/A} = \sum\sum(Y_{ij} - \bar{Y}_{.j})^2 = 16+4+2+2 = 22$	df=N-a=15-3=12	$MS_{S/A} = \frac{SS_{S/A}}{df_{S/A}} = 22/12 = 1.833$	
$SS_T = \sum(Y_{ij} - \bar{Y}_{..})^2 = 52$			

4.10.2. T-Testi

T-testi iki grup veri arasındaki farkı ortalamalarına bakarak tespit eder. Bu farkın gerçeği yansıtıp yansıtmadığı grafiklerle de görülebilir. Başlangıçta iki hipotez oluşturulur. Bunlar H_0 ve H_a olarak adlandırılır. H_0 hipotezi iki grubun farklı olmadığını ortaya atar $H_0: \mu = \mu_0$. Alternatif hipotez H_a doğru ise $H_a: \mu > \mu_0$ durumu geçerlidir. Birinci grubun ortalaması a_{11}, ikinci grubun ortalaması a_{22} ile simgelendiğinde birinci gruptaki örnek sayısı n_1 ve ikinci gruptaki örnek sayısı n_2 dir. Bu ifadelerle S_1^2 *ve* S_1^2 varyansları (Denklem 4.10)'daki gibi hesaplanır. (Denklem 4.11) kullanılarak t-test değeri bulunur.

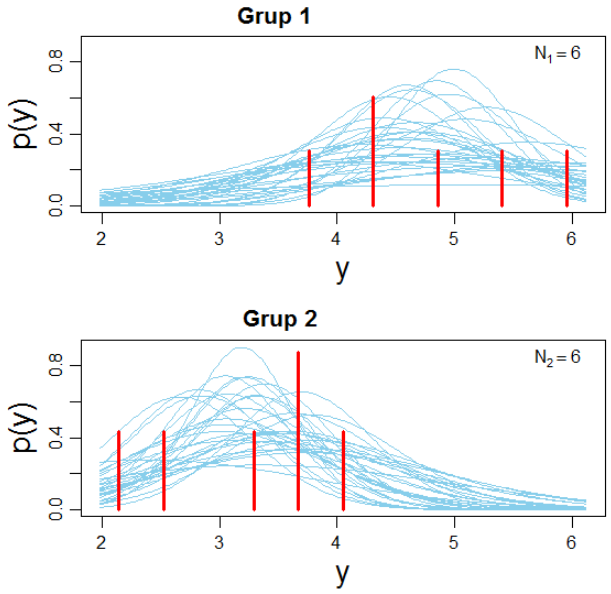
$$S_1^2=(\Sigma(a_1-a_{11})^2/n_1) \qquad S_2^2=(\Sigma(a_2-a_{22})^2/n_2) \tag{4.10}$$

$$t = \frac{\overline{x1} - \overline{x2}}{\sqrt{\frac{s_1^2}{n_1}} + \sqrt{\frac{s_2^2}{n_2}} - \sqrt{2r\left[\frac{s_1}{\sqrt{n_1}}\right]\left[\frac{s_2}{\sqrt{n_2}}\right]}} \tag{4.11}$$

İki grup arasındaki fark Şekil 4.18'deki gibi grafik yardımıyla da görülebilir. Histogram şeklindeki veri dağılımlarında histogramlar arasındaki açıklık arttıkça t-test H_a hipotezi güçlenmiş olur. Aksi durumda yani histogramlar birbirine yakınlaştıkça H_0 hipotezinin geçerliliği kabul edilir. Eşleştirilmiş t-test işleminde ise grup içindeki her gözlem bir diğer gruptaki başka bir gözlem ile ilişkilendirilir.

4.10.3. Ki-kare testi

Ki-kare testinde iki kategorik veri arasındaki ilişki araştırılmaya çalışılır. Frekans farklılıkları burada önemlidir. Örneklerin rastgele alınması ilk adımdır. Her gözlem birbirinden bağımsız olarak yürütülür. Her veri grubunda en az beş veri olmak zorundadır ve veri değeri sıfırdan farklı olmalıdır. Analiz sonucu *p* değeri 0.05 den küçük olanlar arasında frekans farklılığı vardır denebilir. (Denklem 4.12)'de ki-kare testine ait formül verilmiştir.



Şekil 4.18. T-Test grafiği

$$x^2 = \sum \frac{(gozlenenDegerler - beklenenDegerler)^2}{beklenenDegerler} \tag{4.12}$$

Özgürlük derecesi *df* kategorik veri sayısına bağlı olarak değişmektedir. Örneğin normalde -1 olarak alınan *df* iki kategorik veri için 1 olarak alınır. Tablo 4.8'de özgürlük derecesine bağlı olarak *p* değerlerinin sınıflandıırılması görülmektedir.

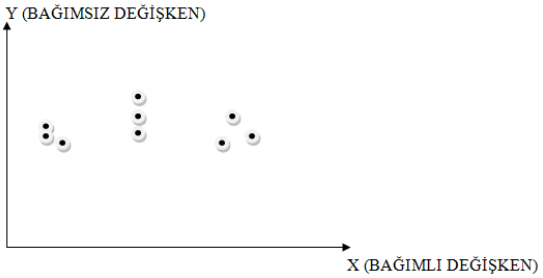
Tablo 4.8. Ki-kare dağılım değerleri

df	**İhtimal (*p*)**								
	0.95	0.9	0.8	0.7	0.5	0.3	0.05	0.01	0.001
1	0.004	0.02	0.06	0.15	0.46	1.07	3.84	6.64	10.83
2	0.1	0.21	0.45	0.71	1.39	2.41	5.99	9.21	13.82
3	0.35	0.58	1.01	1.42	2.37	3.66	7.82	11.34	16.27
4	0.71	1.06	1.65	2.20	3.36	4.88	9.49	13.28	18.47
	Frekans farkı geçersiz						Farklı frekans		

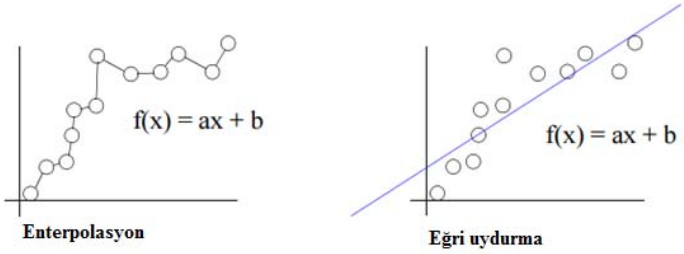
4.10.4. Eğri uydurma

Eğer bir bağımlı değişken bir bağımsız değişkene bağlı olarak tahmin edilecekse "Eğri Uydurma" metodu uygulanabilir [135]. Bu sayede bu değişkenler arasındaki ilişki formulüze edilebilir. En küçük kare gerilemesi (least square regression) ve

enterpolasyon olmak üzere iki tip eğri uydurma mevcuttur. Veriler kesinse ve gürültü oranı azsa enterpolasyon, hata ve gürültü oranının yüksek olduğu durumlarda en küçük kare regresyonu kullanılır. Enterpolasyon noktaların birleştirilmesidir. Ancak noktalar rastgele dağıtılırsa sonuç iyi olmaz. Şekil 4.19'daki veriler için bir *y* fonksiyonu *y=bx+c* ile ifade edildiğinde, *b* ve *c* değerlerini noktalara en iyi uyacak şekilde seçmek amaçlanır. Şekil 4.20'de enterpolasyon ile eğri uydurma arasındaki fark şekil ile gösterilmiştir. Eğri uydurma işleminin başarı ölçütü R^2 ile gösterilir. Bu değerin 1'e yakın olması denklemin kesinliğinin yüksek olduğu anlamına gelmektedir.



Şekil 4.19. Regresyon değerleri



Şekil 4.20. Enterpolasyon-eğri uydurma

Uygunluk fonksiyonu $y=b_0+b_1x$ ile ifade edilirse uygunluk çizgisine en yakın b_0 ve b_1 değerleri seçilir. Her gözlenen cevap yi, tahmin edici xi ile ilişkilendirildiğinde uygun değer $y_i=b_0+b_1x_i$ olur. Böylece karelerin toplamı uzaklıkları en aza indirerek uygun çizgi elde edilmiş olur. Bunun için karelerin toplamının hata oranı *SSE* azaltılmaya çalışılır. (Denklem 4.13)'te *SSE* denklemi verilmiştir.

$$SSE = \sum_{i=1}^{n}(y_i - y_j)^2 = \sum_{i=1}^{n}(y_i - (b_0 + b_1x_i))^2 \qquad (4.13)$$

Eşitlikteki b_0 ve b_1 değerlerini bulabilmek için (Denklem 4.14) ifadeleri kullanılır.

$$b_1 = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sum_{i=1}^{n}(x_i-\bar{x})^2} = \frac{SS_{xy}}{SS_{xx}}$$

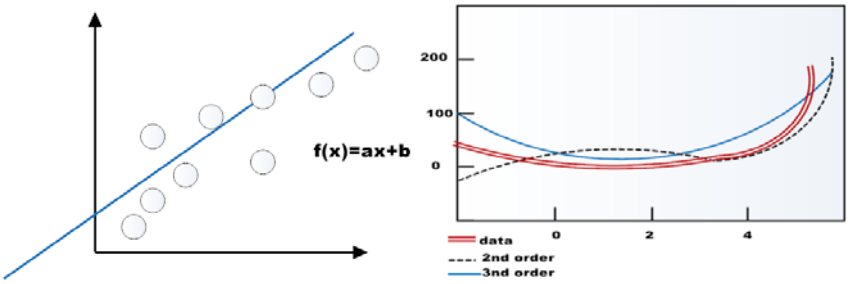
$$b_0 = \bar{y} - \beta_1\bar{x} = \frac{\sum_{i=1}^{n} y_i}{n} - b_1\frac{\sum_{i=1}^{n} x_i}{n} \tag{4.14}$$

x ve y'nin kovaryanslarını hesaplayabilmek için (Denklem 4.15) işlemleri yapılır.

$$SS_{xx} = \sum_{i=1}^{n}(x_i - \bar{x})^2 = \sum_{i=1}^{n}(x_i)^2 - \frac{(\sum_{i=1}^{n} x_i)^2}{n}$$

$$SS_{xy} = \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) = \sum_{i=1}^{n} x_i y_i - \frac{(\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n} \tag{4.15}$$

$$SS_{yy} = \sum_{i=1}^{n}(y_i - \bar{y})^2 = \sum_{i=1}^{n} y_i^2 - \frac{(\sum_{i=1}^{n} y_i)^2}{n}$$



Şekil 4.21. Lineer-Nonlineer eğri uydurma

Şekil 4.21'de lineer-nonlineer eğri uydurma biçimleri gösterilmiştir. Nonlineer eğri uydurma biçiminde denklemler lineer dağılımlara göre daha karmaşıktır. Böyle durumlarda birden fazla bağımsız değişken vardır. Çoklu lineer regresyon olarak da adlandırılan bu durumda (Denklem 4.16)'dakine benzer fonksiyonlar vardır.

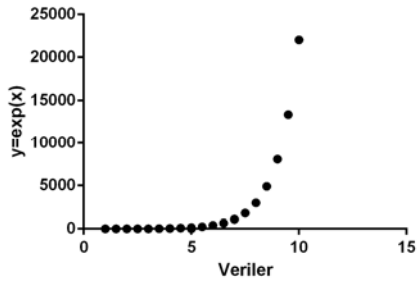
$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \varepsilon \tag{4.16}$$

4.10.5. Spearman analizi

Pearson korelasyonu temelinde geliştirilen Spearman analizi, iki veri arasındaki lineer olmayan ilişkilerin belirlenmesinde kullanılır. Aksine Pearson analizi lineer ilişki seviyesini iki veri arasında ölçer. Eğer verilerimiz normal dağılıma uygun değilse ve lineer olmayan bir ilişki söz konusu ise Spearman analizini kullanmak doğru olacaktır [136]. Spearman analizini kavramak için öncelikle monotonik fonksiyon kavramını anlamak gereklidir. Şekil 4.22'de monotonik fonksiyon çeşitleri görülmektedir. Eğer bir *f(x)* fonksiyonu *x* değerine bağlı olarak artıyorsa burada monotonik artan, *x* değerine bağlı olarak azalıyorsa monotonik azalan, bazen artıp bazen azalıyorsa monotonik olmayan bir ilişki söz konusudur. Spearman analizi katsayısı monotonik ilişkinin seviyesini gösteren bir istatistiksel ölçümdür. Bu ölçüm r_s ile gösterilir. r_s katsayısının -1 ile 1 arasında olması beklenir. Eğer r_s ± 1değerine yakınsa monotonik ilişki güçlüdür denir.

Şekil 4.22. Monotonik fonksiyonlar

Şekil 4.23'te mükemmel bir monotonik fonksiyon örneği verilmiştir. Bu örnekte *y=exp(x)* fonksiyonuna uygun olan çizilen eğri görülmektedir. Monotonik artan ilişkide ilgili *x* değerleri fonksiyona uygulanarak *y* değerleri çizdirilmiştir.



Şekil 4.23. *y=exp(x)* fonksiyonu

BÖLÜM 5. YENİ BİR HİBRİT ÖN-İŞLEME ALGORİTMASI TASARIMI VE HATA TAHMİN ÇERÇEVESİ YAZILIMI

5.1. Ön-işleme Algoritması

Geliştirdiğimiz ön-işleme algoritmasının ismi HSDD (hybrid sampling for defect prediction) dir. Bu ismin verilme nedeni algoritmanın örnekler üzerinde hem çoğaltma hem de azaltma yapabilmesidir. Metrik türetim işlemleri sırasında örnek özellik adedi artarken veri temizleme aşamasında örnek adedi azaltılmaktadır. Şekil 5.1'de algoritma adımları görülmektedir. Algoritma başlangıcında *N* ile veri seti grubu simgelenmektedir. *sm* örnek ortalaması, *gm* ise genel ortalamayı simgelemektedir. Başlangıçta yapılan tTest ve ki-kare testi sonuçları sırasıyla *dt1* ve *dt2* isimli tablolara yüklenmektedir. Eşik değer olarak genel ortalamanın yarısı alınmıştır. HSDD'nin birinci bölümünde *sm* ile *gm* farkı eşik değerden küçük olanlar elenecekler listesine alınmaktadır. Bu işlem *dt1* tablosundaki tüm sonuçlar için tekrarlanmaktadır.

Algoritmanın ikinci bölümünde ise her örneğin frekansı diğer tüm örneklerle karşılaştırılmaktadır. Bu işlem örneklerin ikili kombinasyonları adedince tekrarlanmaktadır. *dt2* tablosundan alınan sonuçlarda her örneğe ait tüm sonuçlardaki *p* değerleri adetleri hesaplanmaktadır. Ortalama *p* değer adedi testi geçemeyen adet ile eşleştirilmektedir. Ortalama adetten küçük olanlar elenerek diğer sonuçlar ayrı bir bölüme eklenerek *Data* tablosu oluşturulmaktadır. tTest sonuçlarından ve ki-kare sonuçlarından gelen elenecek örnekler tek tabloda birleştirilerek yeni sayfaya kaydedilmektedir. Bu işlem veri setlerindeki testler bitinceye kadar tekrar edilmektedir.

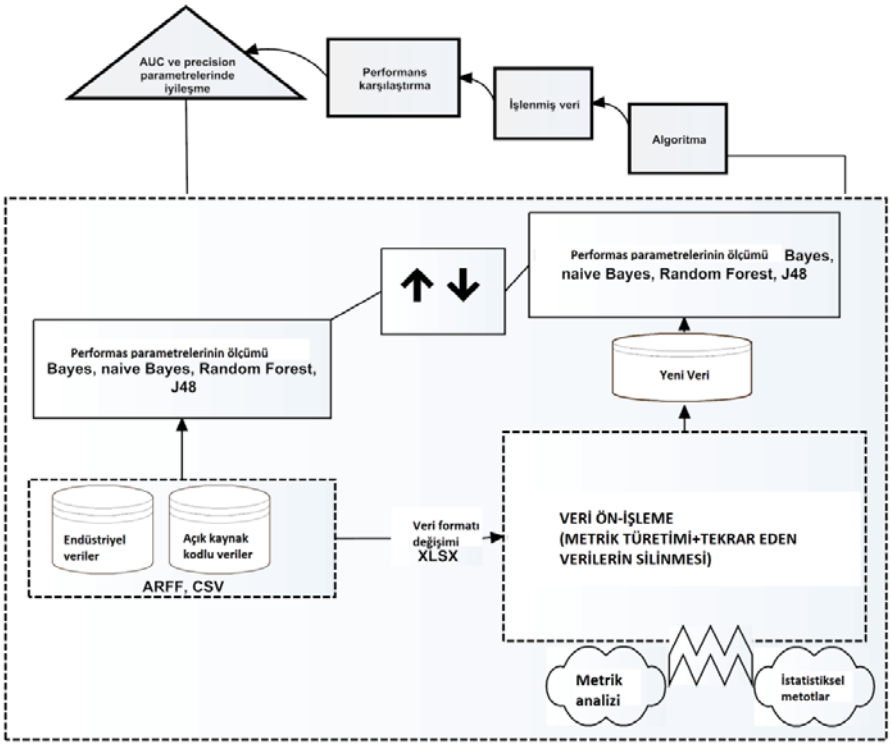
Algoritmanın ikinci bölümünde geçen *dt2[i][k]* ifadesindeki *i* örnek numarasını *k* ise örneğin ilgili özelliğini göstermektedir. Buradaki tüm değerler bir listeye eklenmektedir. İşlenen her liste daha sonra temizlenerek yeni örnek elemanları için hazır hale getirilmektedir. Test uygulanırken dikkat edilmesi gereken noktalardan biri

ki-kare testinin "0" değerli örnekler için uygulanabilir olmamasıdır. Ayrıca böyle istatistiksel analizler belli varsayımlardan hareket ederler. Örneğin tTest ve ki-kare testleri verilerin normal dağıldığını kabul eder. Eğer normal dağılım mevcut değilse bir normal dağılım filtresi yardımıyla veriler teste uygun hale getirilmelidir.

Eğitim veri setlerinde Gaussian dağılıman uygun olmayan eğitim veri setleri için aşağıdaki kod bloğu yazılımıştır.

```
Normalize norm = new Normalize();
norm.setInputFormat(insts);
weka.core.Instances processed_data = Filter.useFilter(insts, norm);
```

Normalizasyon için öncelikle *norm* isimli filtre oluşturulur. Bu filtreye setInputFormat fonksiyonu ile örnekler tanıtılır. Daha sonraki adımda ise useFilter fonksiyonu ile filtreden geçirilen örnekler yeni örnek nesnesine atanır. “0-1” aralığına çekilen tüm değerler Gaussian dağılımına uygun hale getirilmiş olur.



Şekil 5.1. Algoritmanın akışı

Algoritma 1. HSDD

```
1: procedure PRUNE(DATA)
2: N:mylyn, lucene, jdtCore, equinox, eclipsePde, ar1, ar3, ar5, ar6, cm1, jm1, kc1, kc2, kc3, pc1, pc2, pc3, pc4, pc5
3: X_i : Sheet tTest from N; l: row number of tTest; numbers: double array; dt1:load from tTest
4: threshold: gm/2; : dt2: load from chiResult
5: wantedSample=0; sumP=0;count=0;flag=0; list: retrieved sample from chiResult
6: while X_i !=null do
7:       dt: load from X_i
8:       while i ≠ l do
9:       for each k column in dt1 do
10:        add dt1[i][k] to list
11:      end for
12:      numbers← array(list); sub← absolute(sm-gm)
13:      if sub ≤ threshold then
14:      Data← save(numbers)
15:      end if
16:      clear(list)
17:      end while
18:      for each sample in dt2 do
19:           count=0;sumP=0;count=0;flag=0
20:           for each sample in dt2 do
21:                for each column in dt2 do
22:                     add dt2[i][k] to list
23:                end for
24:                numbers ← array(list)
25:                if numbers==wantedSample then
26:                     flag=1
27:                     count++
28:                     add pValues of numbers to smP
29:                end if
30:                clear(list)
31:           end for
32:           if flag==1 then
33:              sumP=sumP/adet
34:              Data← save(numbers)
35:           end if
36:           wantedSample++
37:      end for
38:       return Data
39: end while
40:end procedure
```

Şekil 5.2. HSDD

Şekil 5.1'de HSDD algoritmasına ait grafik özeti görülmektedir. Algoritma dört adımlı

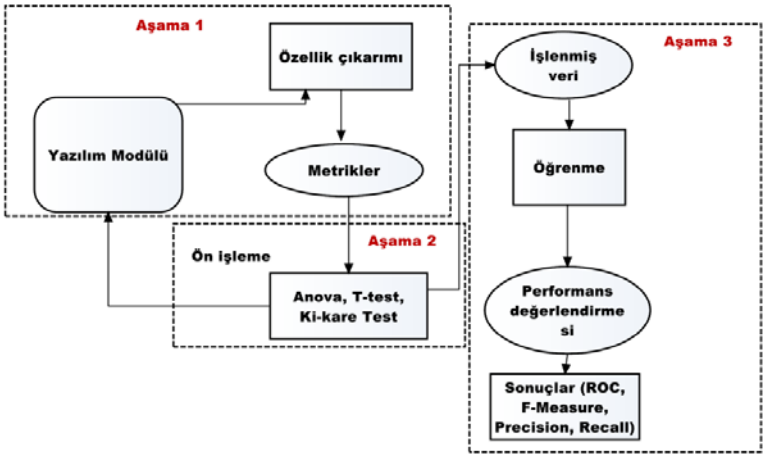
hata tahmin işlemler dizisinin birinci aşamasında gerçekleştirilen metotları içermektedir. Algoritma veriler üzerinde düşük seviyeli metrik türetimi ve tekrar eden verilerin silinmesi işlemlerini xlsx formatına çevrilen arff uzantılı endüstriyel veri setleri üzerinde gerçekleştirmektedir. Buna ek olarak açık kaynak kodlu veri setleri csv formatında oldupu için xlsx formatına dönüştürülmesi ve verilerin yorumlanması xlsx formatı üzerinden yapılmıştır.

HSDD'nin ürettiği yeni veriler işlenmiş veriler olarak adlandırılmakta ve bu veriler üzerinden eski verilerle bir performans karşılaştırılması yapılmaktadır. Performans karşılaştırması için Bayes, Naive Bayes, Rastgele Orman ve J48 algoritmalarının seçilmesinin temel nedeni literatürdeki hata tahmin çalışmalarında sıklıkla görülen sınıflandırma algoritmalarının ağırlıklı olarak bunlardan oluşmasıdır. Aynı şekilde performans değerlendirme parametresi olarak AUC ve Kesinlik seçilmiştir. Bu parametreler de hata tahmininde sıklıkla kullanılmaktadır.

5.2. Yazılım

Yazılım hata tahmini işlemlerini, istatistiksel işlemleri ve metrik işlemlerini içeren, bunun dışında hata veri setleri ile ilgili ön-işleme algoritmasının yer aldığı bir yazılım çerçevesi geliştirilmiştir. Yaklaşımın genel hatları ve içerdiği aşamalar Şekil 5.3'te görülmektedir. Ön-işleme için geliştirilen algoritma HSDD (hybrid sampling for defect data sets) olarak isimlendirilmiştir. Bu ismin verilmesinin temel nedeni algoritmanın hem örnek çoğaltma hem de örnek azaltma fonksiyonunu beraber içermesidir.

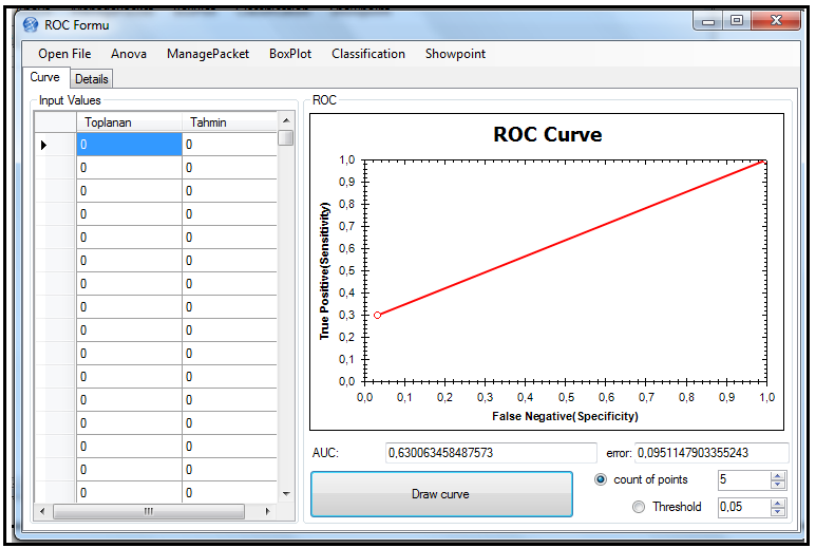
Birinci aşamada yazılım kodlarına ait özellik çıkarımı ve metrik tablolarının oluşturulması gerçekleştirilmiştir. İkinci aşamada ön-işleme için gerekli istatistiksel testler ile metrik türetim ve tekrar eden verilerin silinmesi sağlanmıştır. Son aşamada işlenmiş veriler üzerinde öğrenme algoritmalarının performansı veri işleme öncesi ile bazı performans parametreleri üzerinden karşılaştırılmıştır. Yazılım C# programlama dili kullanılarak Visual Studio 2013 yazılım geliştirme ortamında geliştirilmiştir.



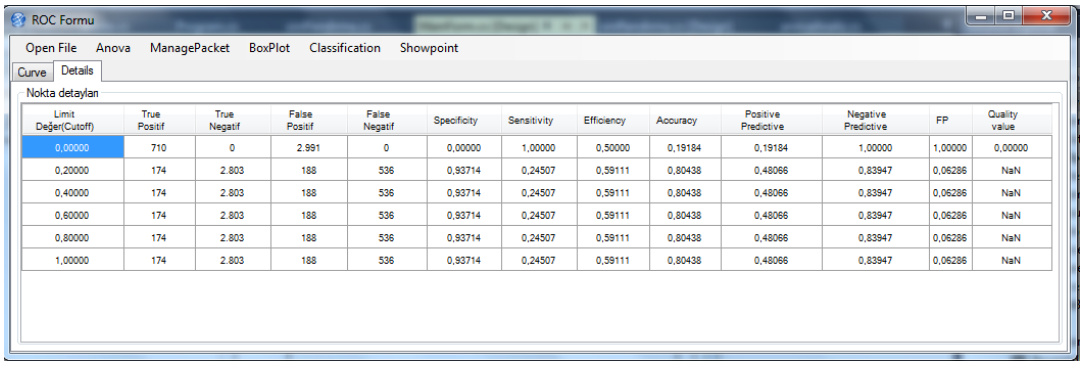
Şekil 5.3. Önerilen çerçevenin işlem aşamaları

Şekil 5.4'te geliştirilen yazılımın ana ekranı görülmektedir. Bu ana ekran üzerinden hata veri setleri üzerinden yapılan tahmin performans parametrelerinden AUC için dosya seçimi yapılabilmektedir. Seçilen dosya 0-1 değerlerinden oluşup ve xlsx formatında olmalıdır. Giriş parametreleri bölümünden seçilen öğrenme algoritmasında yapılan tahmin "Tahmin" sütununu, gerçek hata değeri ise "Toplanan" sütununu oluşturmaktadır. Bu değerler karşılaştırılarak ROC eğrisi ekranın sağ tarafında çizilmektedir. TP değerlerine karşı FN değerlerinin çizilmesiyle oluşturulan eğri kırmızı çizgi ile gösterilmektedir. Eğrinin altındaki bölümde AUC ve hata oranı görülebilmekte, bunun yanında nokta sayısı ve eşik değer belirlenebilmektedir. "Details" bölümünde eğriye ait detaylar yer almaktadır. Örnek detaylar ve oluşturulan detay sütunları Şekil 5.5'te görülmektedir. Detay sütunları arasında limit değer, TP, TN, FP, FN, hassasiyet, özgüllük, etkililik ve kesinlik gibi değerler bulunmaktadır.

Program kullanılarak kod içerisinde çıkabilecek hatalar görüntülenebilmektedir. Düzeltilmesi önerilen hata türlerine ait açıklamalar Tablo 5.1'de görülmektedir. Eğer bir zip dosyası içerisindeki sınıflar listelenirse, seçilen sınıfa bağlı olarak hata üretebilecek satırlar farklı renkte gösterilmektedir. Ayrıca potansiyel hatalı kod satırına ait mesaj bilgisi de aynı form üzerinden verilmektedir. Örnek ekran görüntüsü Şekil 5.6'de verilmektedir.



Şekil 5.4. Yazılım ana ekran

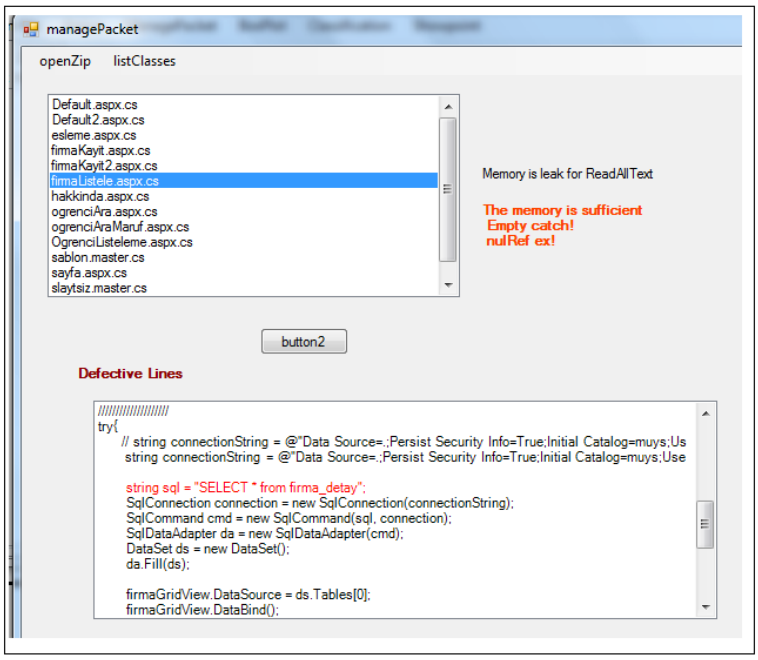


Limit Değer(Cutoff)	True Positif	True Negatif	False Positif	False Negatif	Specificity	Sensitivity	Efficiency	Accuracy	Positive Predictive	Negative Predictive	FP	Quality value
0,00000	710	0	2.991	0	0,00000	1,00000	0,50000	0,19184	0,19184	1,00000	1,00000	0,00000
0,20000	174	2.803	188	536	0,93714	0,24507	0,59111	0,80438	0,48066	0,83947	0,06286	NaN
0,40000	174	2.803	188	536	0,93714	0,24507	0,59111	0,80438	0,48066	0,83947	0,06286	NaN
0,60000	174	2.803	188	536	0,93714	0,24507	0,59111	0,80438	0,48066	0,83947	0,06286	NaN
0,80000	174	2.803	188	536	0,93714	0,24507	0,59111	0,80438	0,48066	0,83947	0,06286	NaN
1,00000	174	2.803	188	536	0,93714	0,24507	0,59111	0,80438	0,48066	0,83947	0,06286	NaN

Şekil 5.5. ROC detayları

Tablo 5.1. Hata türleri

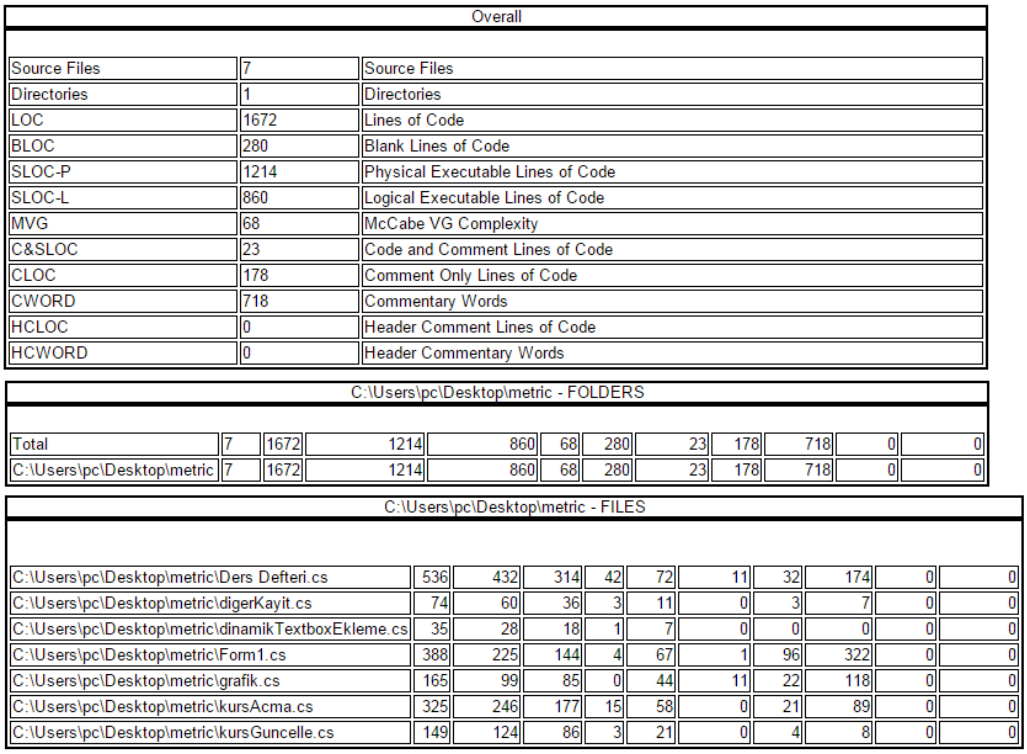
Hata Türü	**Açıklama**
Null pointer	Eğer bir çağrı zinciri içerisinden null bir değer geliyorsa böyle bir hatayla karşılaşılır.
Resource leaks	If source>1000(long)
Boş catch	Blokta hata yakalandığında boş geçilmiş bir mesajın varlığı tespit edilir.
Null int	Int?
String detayı	" " içerisinde yakalanan hataların detaylandırılması



Şekil 5.6. Seçilen sınıfa ait potansiyel hata satırlarının görülmesi

Hata tahmin çerçevesine ait yazılım seçilen klasördeki sınıflara ait metrik tablolarını oluşturabilmektedir. Oluşan özet tablolarda dosya adedi, dizin adedi, satır sayısı, mantıksal satır sayısı, karmaşıklık gibi metriklere ait değerler hesaplanabilmektedir. Ayrıca hangi sınıfta bu metrik değerlerinin nasıl değiştiği de ayrıntılı olarak kaydedilebilmektedir. Şekil 5.7'de örnek metrik hesaplaması ve sınıf ayrıntıları görülmektedir. Üç ana tablo üretilmektedir. Birinci tabloda dosya ve dizin adetleri ile metrikler, ikinci tabloda özet olarak metrik tablosu ve son tabloda sınıflara ait ayrıntılı metrik tablosu üretilebilmektedir.

Burada üretilebilen metrik türleri McCabe ve Halstead standartlarında yer alan tüm metrikleri kapsamamakla birlikte yazılım geliştirme sırasında karşılaşılan hatalara bağlı olarak yazılım modülleri ve sınıflarını değerlendirmeye yardımcı olmaktadır. Tera-promise veri ambarından alınarak yapılan hata tahmin çalışmalarının geçerliliğini test edebilmek daha farklı veri setleri ile çalışmak gerekir. Dolayısıyla yazılım kodları üzerinde inceleme yaparak metrik tabloları üretebilen yazılımlara olan ihtiyaç günden güne artmaktadır. Ayrıca bu yazılımların çeşitlenmesi bilimsel bulguların genelliğinin onaylanabilmesi için önemlidir.



Overall

Source Files	7	Source Files
Directories	1	Directories
LOC	1672	Lines of Code
BLOC	280	Blank Lines of Code
SLOC-P	1214	Physical Executable Lines of Code
SLOC-L	860	Logical Executable Lines of Code
MVG	68	McCabe VG Complexity
C&SLOC	23	Code and Comment Lines of Code
CLOC	178	Comment Only Lines of Code
CWORD	718	Commentary Words
HCLOC	0	Header Comment Lines of Code
HCWORD	0	Header Commentary Words

C:\Users\pc\Desktop\metric - FOLDERS

Total	7	1672	1214	860	68	280	23	178	718	0	0
C:\Users\pc\Desktop\metric	7	1672	1214	860	68	280	23	178	718	0	0

C:\Users\pc\Desktop\metric - FILES

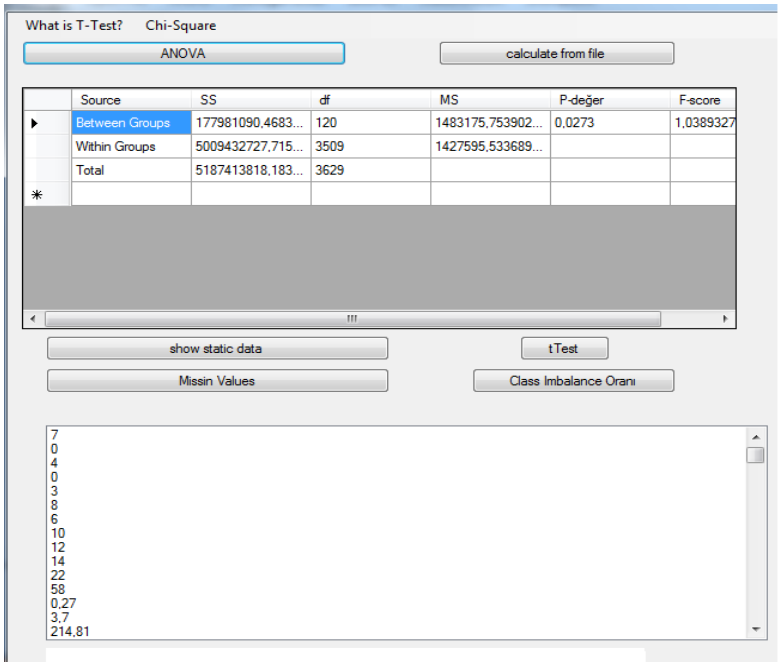
C:\Users\pc\Desktop\metric\Ders Defteri.cs	536	432	314	42	72	11	32	174	0	0
C:\Users\pc\Desktop\metric\digerKayit.cs	74	60	36	3	11	0	3	7	0	0
C:\Users\pc\Desktop\metric\dinamikTextboxEkleme.cs	35	28	18	1	7	0	0	0	0	0
C:\Users\pc\Desktop\metric\Form1.cs	388	225	144	4	67	1	96	322	0	0
C:\Users\pc\Desktop\metric\grafik.cs	165	99	85	0	44	11	22	118	0	0
C:\Users\pc\Desktop\metric\kursAcma.cs	325	246	177	15	58	0	21	89	0	0
C:\Users\pc\Desktop\metric\kursGuncelle.cs	149	124	86	3	21	0	4	8	0	0

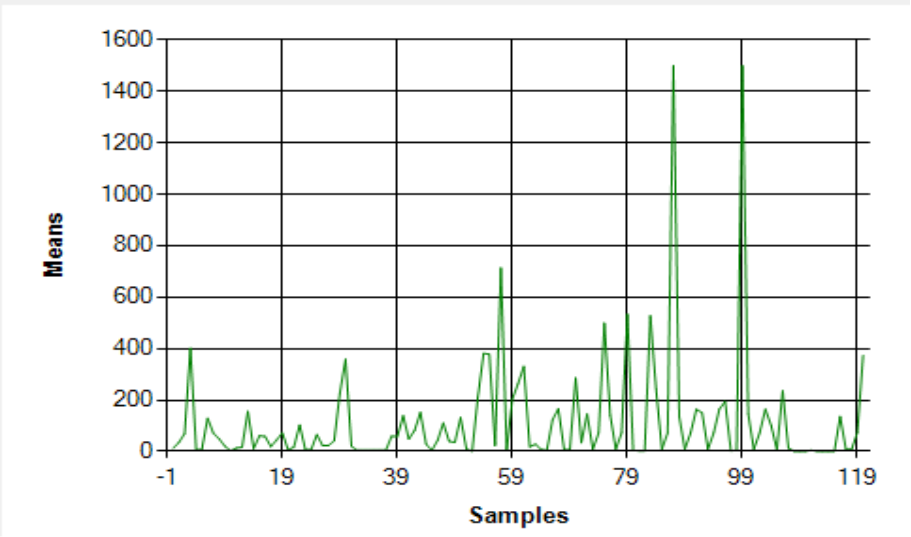
Şekil 5.7. Seçilen kodlara ait metrik detayları

Metrik üretim fonksiyonu yazılım içerisinde yer almakla birlikte deneysel veri setlerinin üretiminde kullanılmamış olup, daha sonra yapılacak veri seti üretim projelerinde kullanılması amaçlanarak hazırlanmış bir fonksiyondur. Orta doğu bölgesi ve Asya'da geliştirilen yazılımlara ait bir hata veri seti ambarı için metrik tablosu üretim fonksiyonu kullanılabilir.

Şekil 5.8'de ANOVA hesaplamalarına ait ekran görülmektedir. Bu form üzerinden ANOVA tablosu oluşturulabilmekte, metrik değerleri görüntülenmekte ve örneklere ait ortalama değerleri çizdirilebilmektedir. Dengesiz veri dağılımı hesaplamalarında kullanılan ortalama hatalı örnek adetlerinin bulunduğu kısımlar rahatlıkla bu grafik sayesinde tespit edilebilmektedir.

Hata veri setleri üzerinde yapılabilecek ön-işleme ve sınıflandırma işlemleri Şekil 5.9'daki form aracılığı ile yapılabilmektedir. Bu form üzerinden yapılacak ön-işleme için veri seti adı girilmektedir. Veri setleri arff uzantılı olarak debug klasörü altında yer almalıdır.



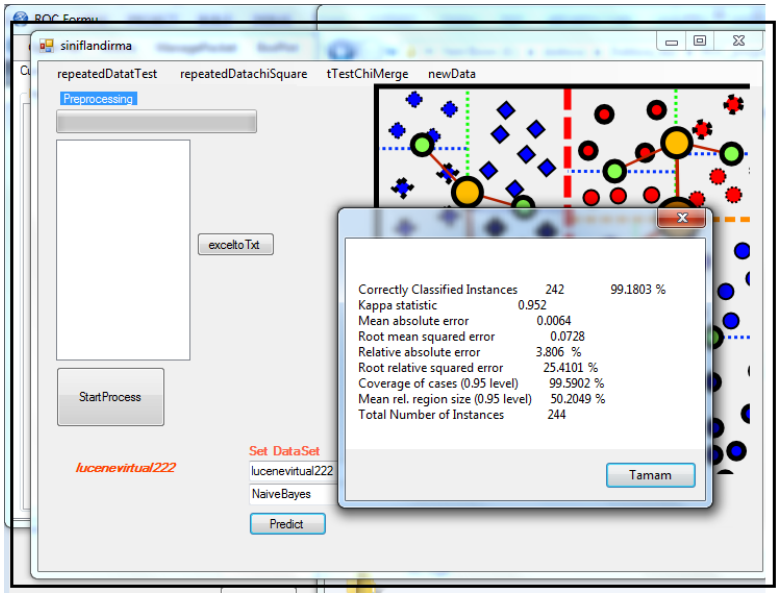


Şekil 5.8. ANOVA analiz ekranı

Dört sınıflandırıcı Bayes, Naive Bayes, Rastgele Orman ve J48 seçilebilmektedir. Bu sınıflandırıcılara ait performans parametreleri işlemler sonrası görüntülenmektedir. Ayrıca dört sınıflandırıcıya ait g-ortalama performans parametresinin ortalaması da bu form ile hesaplanabilmektedir. weka.Core kütüphanesi kullanılarak yapılan sınıflandırma ve diğer işlemler WEKA programı ile tekrar edilerek karşılaştırılabilmektedir. Programdaki deney ortamı 10*10 çapraz-onaylama ayarlarına

sahiptir. Formun sol üst bölümünde seçilen veri setine ait t-test ve ki-kare testi işlemleri yapılabilmekte ve bu test sonuçları xlsx formatında sonuç raporuna birleştirilerek kaydedilebilmektedir.

Yazılım çerçevesi txt, arff ve xlsx formatındaki veriler ile işlem yapmaktadır. Farklı formatlarda hata tahmin verilerinin işlenmesinin nedeni istatistiksel analiz yöntemleri ile tahmin yöntemlerinin birbirinden farklı olmasıdır. Örneğin hata veri setlerinde yapılacak örnek azaltma veya çoğaltma işlemleri txt formatında daha hızlı gerçekleşmektedir. Diğer taraftan verilerin incelenmesi, ROC analizi ve düzenli ifadesi xlsx formatında daha uygundur. Sınıflandırma işlemlerinin yapıldığı weka.Core kütüphanesi arff uzantılı verilerle çalışmaktadır. Dolayısıyla bu formatta da hata veri setlerimiz mevcuttur.



Şekil 5.9. Sınıflandırma ve ön-işleme ekranı

İlk t-test analizinde genel ortalamadan farklı olmayan örnekler satır numaraları ve ortalama değerleri ile kaydedilmektedir. Şekil 5.10'de lucene veri setine ait t-test işlemlerinin ilk adımı görülmektedir. "False" sonucuna sahip örnekler genel dağılımdan farklı olmayan örnekler sınıfına girmektedir.

	A	B	C	D	E
1	satirNo	significant	ort	ortOrt	
2	0	False	241,391304347826	152,006858365318	
3	1	False	1256,95652173913	152,006858365318	
4	2	False	217,434782608696	152,006858365318	
5	3	False	57,8695652173913	152,006858365318	
6	5	False	367,521739130435	152,006858365318	
7	7	False	474	152,006858365318	
8	8	False	65,8260869565217	152,006858365318	
9	9	False	268,217391304348	152,006858365318	
10	10	False	260,434782608696	152,006858365318	
11	14	False	57,8260869565217	152,006858365318	
12	16	False	95,6086956521739	152,006858365318	
13	17	False	123,04347826087	152,006858365318	
14	18	False	80,1739130434783	152,006858365318	
15	19	False	105,95652173913	152,006858365318	
16	20	False	59,4782608695652	152,006858365318	
17	22	False	155,608695652174	152,006858365318	
18	23	False	62,695652173913	152,006858365318	
19	24	False	79,0434782608696	152,006858365318	

Şekil 5.10. Birinci adım t-test

Birinci adım ki-kare testinde her örnek diğer tüm örnekler adedince teste tabi tutulur. Sonucu "False" olan test sonuçları ikili satır numaraları ve p-değer büyükükleri ile kaydedilir. Bu işlem tüm verilerin ikili kombinasyonları adedince tekrarlanır.

Şekil 5.11'da ki-kare testine ait birinci adımda elde edilen test sonuç tablosunun küçük bir bölümü görülmektedir. Dikkat edilirse 3 ve 4 numaralı örneklere ait test sonuçları görülmektedir. 1 ve 2 numaralı örneklere ait sonuçlar "True" olduğu için alınmamıştır. Ayrıca tüm örneklerde 3. örneğe ait 8 adet "False" sonucu bulunabilmiştir. p-değerlerinin 0.05 değerinin üstünde olduğu da görülmektedir. Zaten bu eşik değerin üstündeki sonuçlar "False" olarak işaretlenmektedir.

Şekil 5.12'de t-test işleminin ikinci ve üçüncü adımları görülmektedir. Bu aşamalarda iki kez eşik değer oluşturularak eşik değerin altında kalan test sonuçları ayrı bir sayfada kaydedilebilmektedir. İşlemin iki kez tekrarlanmasının nedeni programın elenecek verileri doğru tespit edebilmesini sağlamaktır.

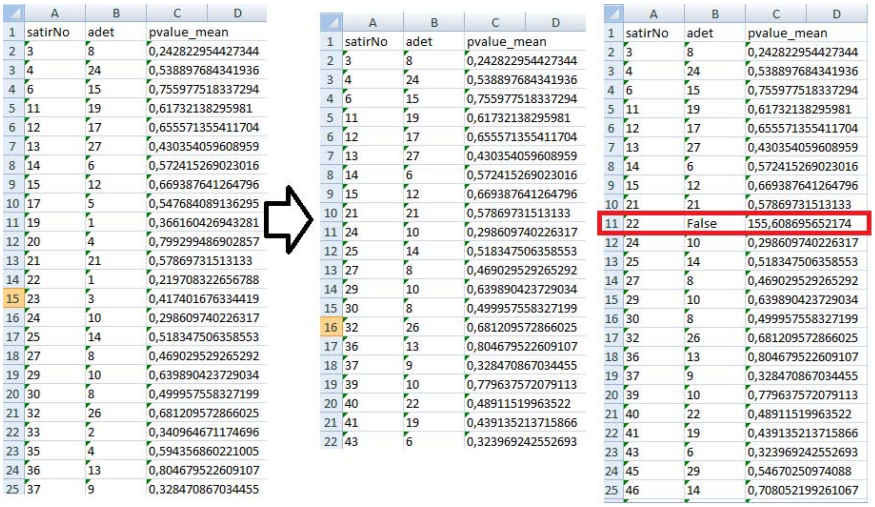
	satir1	satir2	significance	pValue
2	3	23	False	0,40850504960407
3	3	52	False	0,38937160898104
4	3	74	False	0,177946842343979
5	3	322	False	0,352631206754571
6	3	514	False	0,128453018698122
7	3	542	False	0,0661666381574705
8	3	589	False	0,286958784145467
9	3	677	False	0,132550486734035
10	4	32	False	0,992477791907014
11	4	40	False	0,999551716684392
12	4	47	False	0,278825471094172
13	4	60	False	0,635871981551023
14	4	71	False	0,999563681590036
15	4	116	False	0,924578861995136
16	4	168	False	0,0923387158538414
17	4	194	False	0,992477791907014
18	4	234	False	0,840727456447846
19	4	255	False	0,0501840931395769
20	4	264	False	0,176310572554027
21	4	287	False	0,935941046479297

Şekil 5.11. Birinci adım ki-kare

	satirNo	significant	ort
2	2	False	217,434782608696
3	16	False	95,6086956521739
4	17	False	123,04347826087
5	18	False	80,1739130434783
6	19	False	105,95652173913
7	22	False	155,608695652174
8	24	False	79,0434782608696
9	26	False	112,04347826087
10	28	False	105,739130434783
11	30	False	79,9130434782609
12	31	False	181,217391304348
13	34	False	205
14	38	False	124,434782608696
15	50	False	110
16	56	False	91,7826086956522
17	63	False	83,0434782608696
18	66	False	159,086956521739
19	80	False	150,173913043478
20	84	False	122,826086956522
21	89	False	95,3913043478261
22	90	False	226,347826086957

	A	B	C	D
1	satirNo	significant	ort	
2	22	False	155,608695652174	
3	80	False	150,173913043478	
4	123	False	156,260869565217	
5	131	False	150,130434782609	
6	373	False	147,521739130435	

Şekil 5.12. İkinci ve üçüncü adım t-test



	A	B	C	D
1	satirNo	adet	pvalue_mean	
2	3	8	0,242822954427344	
3	4	24	0,538897684341936	
4	6	15	0,755977518337294	
5	11	19	0,61732138295981	
6	12	17	0,655571355411704	
7	13	27	0,430354059608959	
8	14	6	0,572415269023016	
9	15	12	0,669387641264796	
10	17	5	0,547684089136295	
11	19	1	0,366160426943281	
12	20	4	0,799299486902857	
13	21	21	0,57869731513133	
14	22	1	0,219708322656788	
15	23	3	0,417401676334419	
16	24	10	0,298609740226317	
17	25	14	0,518347506358553	
18	27	8	0,469029529265292	
19	29	10	0,639890423729034	
20	30	8	0,499957558327199	
21	32	26	0,681209572866025	
22	33	2	0,340964671174696	
23	35	4	0,594356860221005	
24	36	13	0,804679522609107	
25	37	9	0,328470867034455	

	A	B	C	D
1	satirNo	adet	pvalue_mean	
2	3	8	0,242822954427344	
3	4	24	0,538897684341936	
4	6	15	0,755977518337294	
5	11	19	0,61732138295981	
6	12	17	0,655571355411704	
7	13	27	0,430354059608959	
8	14	6	0,572415269023016	
9	15	12	0,669387641264796	
10	21	21	0,57869731513133	
11	24	10	0,298609740226317	
12	25	14	0,518347506358553	
13	27	8	0,469029529265292	
14	29	10	0,639890423729034	
15	30	8	0,499957558327199	
16	32	26	0,681209572866025	
17	36	13	0,804679522609107	
18	37	9	0,328470867034455	
19	39	10	0,779637572079113	
20	40	22	0,48911519963522	
21	41	19	0,439135213715866	
22	43	6	0,323969242552693	

	A	B	C	D
1	satirNo	adet	pvalue_mean	
2	3	8	0,242822954427344	
3	4	24	0,538897684341936	
4	6	15	0,755977518337294	
5	11	19	0,61732138295981	
6	12	17	0,655571355411704	
7	13	27	0,430354059608959	
8	14	6	0,572415269023016	
9	15	12	0,669387641264796	
10	21	21	0,57869731513133	
11	22	False	155,608695652174	
12	24	10	0,298609740226317	
13	25	14	0,518347506358553	
14	27	8	0,469029529265292	
15	29	10	0,639890423729034	
16	30	8	0,499957558327199	
17	32	26	0,681209572866025	
18	36	13	0,804679522609107	
19	37	9	0,328470867034455	
20	39	10	0,779637572079113	
21	40	22	0,48911519963522	
22	41	19	0,439135213715866	
23	43	6	0,323969242552693	
24	45	29	0,54670250974088	
25	46	14	0,708052199261067	

Şekil 5.13. İkinci ve üçüncü adım ki-kare

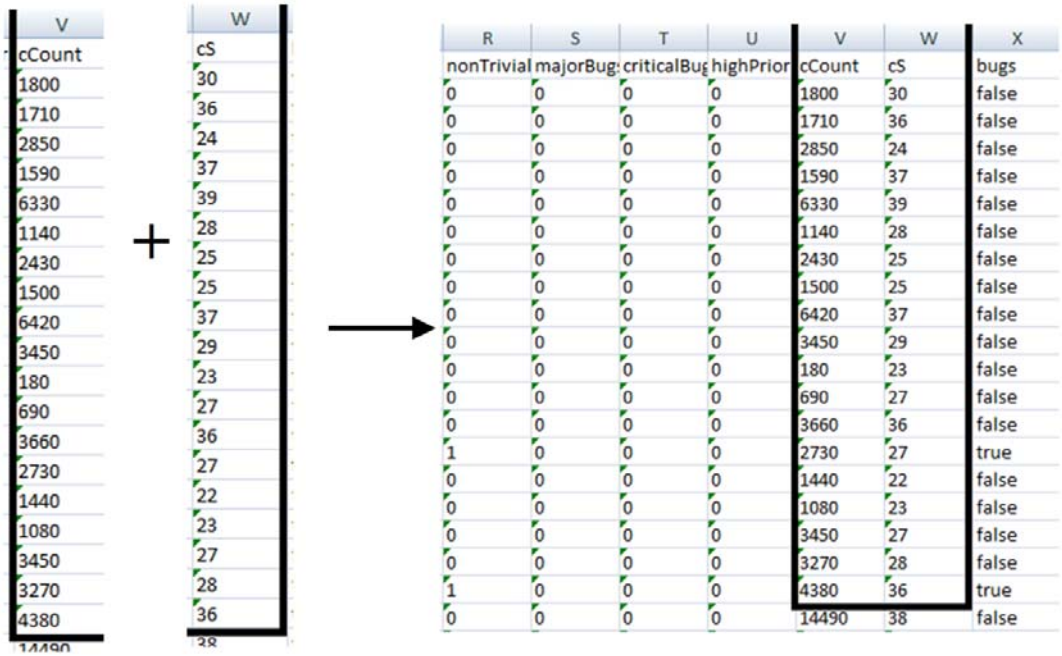
Ki-kare testinin ikinci aşamasında hangi örneğe ait kaç adet "False" sonuçlu testin olduğu hesaplanıyor. Üçüncü aşamada ise ortalama adetten küçük olan sonuçlar elenerek ortalamadan büyük adede sahip örnekler elenecek örnekler sayfasına ekleniyor. Son olarak yazılım iki ayrı istatistiksel analizden gelen sonuçları tek sayfada birleştiriyor. Şekil 5.13'deki dikdörtgen içindeki örnek birleştirilen sonuçlardaki t-test tablosundan gelen bir bölümdür.

5.3. Metrik Türetimi

Metrik türetiminin temel amacı yazılım modüllerinin daha iyi ayırt edilebilmesidir. Böylece düşük seviyeli metriklerin sınıflandırma başarısındaki etkisi bulunacaktır. Daha detaylı metrikler tekrar eden veri noktalarının belirlenmesine yardımcı olacak ve tekrar edilen veriler elendikten sonraki sonuçlar gözlemlenecektir. İki adet düşük seviyeli metrik türetimi gerçekleştirilmiştir. Bunlar class size (cS) ve karakter sayısı (cCount) metrikleridir. cS metriği Lorenz and Kidd metrik standartları arasında var olan düşük seviyeli bir metriktir. cCount metriği ise metrik standartlarında yer almayan ancak bilgi verici düşük seviyeli bir metriktir. Bu metriklerin ikisi de deneysel çalışmada kullandığımız veri setlerinden yer almayan sonradan eklenen metriklerdir. Hata tahmininde kullanılan veri setlerinin tamamının açık kaynak kodlu olmaması metrik türetimi işlemini zorlaştırmaktadır. Ancak böyle metrikler regresyon analizleri yardımıyla tablolardaki eski metriklerden türetilmesi daha kolay düşük seviyeli

metriklerdir. Ayrıca daha önceki hata tahmin çalışmalarında özellikle cCount metriğinin kullanımı cesaretlendirilmiştir. 5.1 numaralı denklem deneysel veri setlerinde yapılan regresyon analizleri sonucunda ortaya çıkarılmıştır. Eşitlikteki cS, Lorenz and Kidd standartlarında yer alan sınıf büyüklüğü metriğidir (total number of operations(tnoo), number the attributes (nta)).

$$cCount \cong lCode$$
$$cS = tnoo + nta \qquad (5.1)$$



Şekil 5.14. Metrik türetimi

Şekil 5.14'te metrik türetimi sonucu oluşan yeni metrik sütunları ve bunlarım veri setlerine eklenmesi işleminin akışını görmekteyiz. Türetilen metrikler hata etiketi sütunundan önce eklenmiştir. Türetilen metriklerin en önemli özelliği birbirleriyle ilişki parametreler içermemeleridir. Dolayısıyla bu metrikler ile yapılan analizle ve değerlendirmeler birbirinden bağımsız ele alınmalıdır. Aynı şekilde yapılan regresyon analizlerinin daha ayırt edici sonuçlar vermesik bu yolla sağlanmıştır.

BÖLÜM 6. DENEYLER

6.1. Veri Seti Detayları

Deneysel veri setlerine ait detaylar Tablo 6.1'de görülmektedir. Deneysel tasarımda kullanılan açık kaynak kodlu veri setleri dokuz farklı sürümlere sahiptir. Biz deneysel tasarım için sekizinci veri setini seçtik. Tablo 6.2'deki Chidamber-Kemerer standartlarını kullanan veriler Nasa MDP veri seti metriklerine daha uyumludur. Deneysel veri setleri C, C++ ve Java programlama dilleri ile yazılan projelerden elde edilmiştir. 15 endüstriyel ve 5 açık kaynak kodlu projenin seçilmesinin nedeni sonuçların genelliğini açık kaynak kodlu projelerde de test edebilmek içindir. Tablo 6.1'deki verilerin hatalı örnek oranları "Hatalı örnek%" sütunu ile gösterilmiştir. "Tip" sütununda proje tipleri hakkında açıklamalar yer almaktadır. "Özellik" sütunu incelendiğinde 21 ile 40 arasında değişen metrik adedinin projelerde değişkenlik gösterdiği görülmektedir. Bu durum ön-işleme ortamının her veri setine göre ayarlanmasından dolayı işlemlerin bitiş zamanlarını arttırmıştır.

Tablo 6.1. Deneysel veri setlerinin detayları

İsim	Dil	Özellik	Örnek sayısı	Hatalı örnek%	Tip
ar1	C	29	241	8	Ticari
ar3	C	29	125	12	Ticari
ar4	C	29	213	18	Ticari
ar5	C	29	71	21	Ticari
ar6	C	29	201	14	Ticari
cm1	C	40	327	12	Nasa MDP
jm1	C	21	10878	19	Nasa MDP
kc1	C++	21	2107	15	Nasa MDP
kc2	C++	21	521	20	Nasa MDP
kc3	Java	40	458	9	Nasa MDP
pc1	C	40	1107	7	Nasa MDP
pc2	C	40	5589	0.4	Nasa MDP
pc3	C	40	1563	10	Nasa MDP
pc4	C	40	1458	12	Nasa MDP
pc5	C++	39	17186	3	Nasa MDP
Eclipse JDT Core	Java	21	997	20	Açık kaynak
Eclipse PDE UI	Java	21	1497	12	Açık kaynak
Equinox Framework	Java	21	324	39	Açık kaynak
Lucene	Java	21	691	9	Açık kaynak
Mylyn	Java	21	1862	13	Açık kaynak

Deneysel veri setleri iki farklı türü içermektedir. Bunlar endüstriyel veri setleri ve açık kaynak kodlu veri setleridir. Endüstriyel veri setleri 15 adet veri seti olup tera-promise veri ambarında bulunan NASA'nın uzay enstrümanlarının yazılımlarından elde edilen veri setleri ile birlikte Boğaziçi yazılım laboratuvarında oluşturulan ar kodlu veri setlerini de içermektedir. Bu veri setlerinin özellik adetleri 21 ile 40 arasında değişmektedir. 15 endüstriyel veri seti McCabe ve Halstead standartları kullanılarak oluşturulan veri setleridir. Tablo 6.2 ve 6.3'te açık kaynak kodlu ve endüstriyel veri setlerine ait metrikler görülmektedir.

Bu veri setlerindeki örnek adetleri ve özellik adetleri proje tiplerine bağlı olarak farklılık göstermektedir. Örneğin ar kodlu projelerdeki özellik sayısı birbirinin aynıyken bu durum pc kodlu projelerde de böyledir. Ancak farklı kodlara sahip veri setlerinde hem özellik sayıları hem de örnek sayıları değişmektedir. Bu durum deneysel tasarımın sonuçlarının veri setleri için ayrı ayrı değerlendirilmesini gerektirmektedir. Diğer taraftan ön-işleme ile ilgili işlemlerin yürütülmesinde bu farklılıklardan kaynaklanan sorunlar ortaya çıkmıştır. Her deney veri seti için yazılım ortamında ayarlamalar yapılmıştır.

Açık kaynak kodlu veri setlerinde ise projeler birbirinden farklı olmasına rağmen özellik sayısı sabittir. Dolayısıyla deney gerçekleştirim zamanı açık kaynak kodlu projelerde endüstriyel veri setleri ile kıyaslandığında daha kısadır.

Açık kaynak kodlu veri setleri 5 farklı projeye ait 91-97 arasında değişen sürüm adedi içermektedir. Her projeye ait 8 farklı veri seti bulunmaktadır. Ancak biz bu veri setlerinden her proje için hata tahminine en uygun olan 8. veri setini seçtik. Seçtiğimiz veri seti Chidamber-Kemerer standartlarını kullanan metrikleri içermektedir.

6.2. Test Detayları

Tablo 6.4'te ön-işleme algoritmasındaki tTest sonuçları görülmektedir. Testin her bir sonucu iki değerlidir true/false. Bu değerlerden "False" olanlar test sonucunu geçemeyen örnekler olup elenebilecek örnekler olarak değerlendirilmektedir. "tTest" sütununda "False" sonuçlu örnek adetleri yer almaktadır. "AşamaI" de genel

ortalamanın yarısı eşik değer olarak alınarak yapılan test sonucunda elenecek örnek adetleri kayda değer şekilde azaltılmıştır. Testin geçerliliği için "Aşama II" de test tekrarlanarak kesin olarak elenecek örnekler belirlenmektedir.

Tablo 6.2. Açık kaynak kodlu veri setlerine ait metrik detayları

cbo	Bir sınıfta tanımlanan metodun veya örneğin diğer bir sınıfta kullanılmasıdır. Bu ilişkinin iki yönlü mü yada tek yönlü mü olduğuna bakılmaksızın tek sayım yapılır. *Aynı sınıfa çoklu erişimde de tek sayım yapılır
dit	Maksimum miras yolu
fanIn	Procedure bağımlı olan procedur sayısı *İstenen bir değer fakat makul olmalı
fanOut	Procedurun bağımlı olduğu procedur sayısı *Bu değer ne kadar fazlaysa kodun test edilmesi ve çalıştırılması o kadar zordur denebilir.
lcom	Metot ile yerel değişkenler arasındaki ilişkinin ölçümüdür.
noc	Temel sınıftan türetilmiş sınıf adedi
numberOfAttributes	Ortalama özellik sayısı
numberOfAttributesInherited	Miras alınan özellik sayısı
numberOfLinesOfCode	Satır sayısı
numberOfMethods	Metot sayısı
numberOfMethodsInherited	Miras alınan metod sayısı
numberOfPrivateAttributes	private Ortalama özellik sayısı
numberOfPrivateMethods	private Metot sayısı
numberOfPublicAttributes	public Ortalama özellik sayısı
numberOfPublicMethods	public Method sayısı
rfc	Sınıf için cevap sayısı
wmc	Bir sınıftaki tüm metotların karmaşıklıkları toplamı
nonTrivialBugs	Önemsiz olmayan hara sayısı
majorBugs	Ana hatalar
criticalBugs	Kritik hatalar
highPriorityBugs	Yüksek öncelikli hatalar
bugs	Hata adedi

Tablo 6.3. Endüstriyel veri setlerine ait metrik detayları

total_loc	Toplam satır sayısı
blank_loc	Toplam boş satır sayısı
comment_loc	Yorum satır sayısı
code_and_comment_loc	Kod+yorum satır sayısı
executable_loc	Çalıştırılabilir satır sayısı
unique_operands	Özgün işenen sayısı
unique_operators	Özgün işleç sayısı
total_operands	Toplam işlenen sayısı (n2)
total_operators	Toplam işleç sayısı (n1)
halstead_vocabulary	İşleç+işlenen sayısı (n)
halstead_length	Toplam işleç+işlenen sayısı (N)
halstead_volume	$N * \log_2 n$
halstead_level	$\frac{2 * n_2}{n_1 * N_2}$
halstead_difficulty	D=$\frac{n_1}{2} * N_2/n_2$
formal_parameters	D*V
halstead_effort	$\frac{E^{\frac{2}{3}}}{3000}$
halstead_error	Program yazımı için gerekli zaman
halstead_time	Dallanma adedi
branch_count	Karar adedi
decision_count	Çağrı çifti
call_pairs	Durum adedi
condition_count	Çoklu durum adedi
multiple_condition_count	Karmaşıklık
cyclomatic_complexity	Yoğunluk
cyclomatic_density	Karar yoğunluğu
decision_density	Tasarım karmaşıklığı
design_complexity	Tasarım yoğunluğu
design_density	normalleşmiş karmaşıklık
normalized_cyclomatic_complexity	

Tablo 6.4'teki analiz gözlenen değerlerle beklenen değerler arasındaki frekans farklılığını karşılaştırmak için kullanılır. Böylece veri setlerindeki frekans farklılıkları gösterilmiş olur. Aynı zamanda bu analiz deneysel verilerin genel hata dağılımını temsil edip etmediğini belirler. Sonuçlarda elde edilen p değerinin $p<0.005$ olması beklenir. Deney sırasındaki kısıtlardan biri verilerin tamamının

sıfırdan farklı olması gerekliliğidir. Bu yüzden bazı metrikler analize dahil edilememiştir.

Tablo 6.4. T-test sonuçları

Veri Seti	Orjinal örnek sayısı	tTest (False)	Aşama I	Aşama II
mylyn	1862	1070	583	34
lucene	691	363	186	5
jdtCore	997	483	248	27
equinox	324	166	88	13
eclipsePde	1497	937	501	36
ar1	121	56	35	9
ar3	63	42	20	4
ar4	107	48	26	12
ar5	36	11	10	10
ar6	101	51	22	18
cm1	327	112	43	10
jm1	10885	2588	1193	1317
kc1	2109	600	237	164
kc2	522	146	68	13
kc3	458	144	57	43
pc1	10885	2588	1193	506
pc2	5589	866	330	265
pc3	1563	413	203	41
pc4	1458	520	242	91
pc5	17186	897	346	156

Örneğin code_and_comment_loc metriği sıfır değeri içerdiği için deney ortamında yer almamıştır. Ki-kare testi örneklerin ikili kombinasyonu adedince tekrarlanmaktadır. Her örnek diğer örneklerle ki-kare testine sokulur. "False" etiketli sonuçlar kaydedilir. "True" etiketli sonuçlar ise frekans farklılığının kayda değer olduğunu göstermektedir. Birinci aşamada "False" etkiketli sonuçların her örnek için adedi hesaplanır. İkinci aşamada ise ortalama "False" adedinden büyük olan örnekler elenecekler listesine eklenir.

Tablo 6.4 ve Tablo 6.5'teki elenecekler listeleri tek tabloda birleştirilir. Sonrasında tüm veri setlerinden elenecek örnekler silinir. Bu analizlerde dikkat çekici nokta tTest ve ki-kare testlerinden elde edilen silinecek örnekler birbirinden farklıdır. Bu da iki analizin verilerin farklı özelliklerini kontrol ettiğini göstermektedir.

Tablo 6.5. Ki-kare testi sonuçları

Veri Seti	Orjinal örnek sayısı	Test adedi	Aşama I	Aşama II
mylyn	1862	3463321	18783	413
lucene	691	476100	371	163
jdtCore	997	992016	379	140
equinox	324	104329	154	55
eclipsePde	1497	2238016	7211	281
ar1	121	14400	41	14
ar3	63	3844	6	6
ar4	107	11236	13	13
ar5	36	1225	1	1
ar6	101	10000	113	16
cm1	327	106276	27	25
jm1	10885	118461456	65535	114
kc1	2109	4443664	65535	244
kc2	522	272484	7768	98
kc3	458	208849	3880	101
pc1	10885	118461456	65353	114
pc2	5589	31225744	34577	25
pc3	1563	2439844	7134	256
pc4	1458	7044	750	267
pc5	17186	65535	1	1

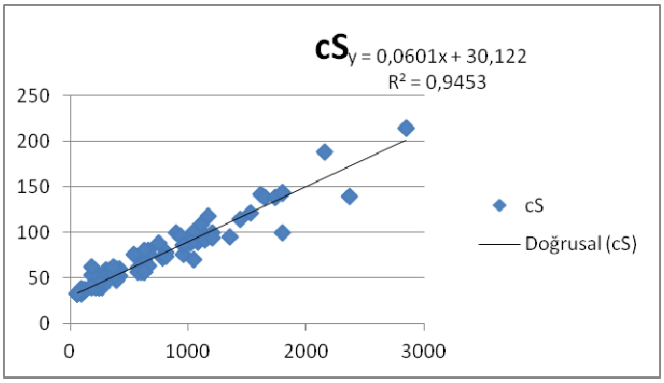
Tablo 6.6 tekrar eden veri noktalarının temizlenmesi işlemine ait detayları sunar. Bu detaylar tTest ve ki-kare testlerinden gelen elenecek veri oranlarını kapsar. Tekrar eden veriler bu tabloya göre endüstriyel verilerde %17, açık kaynak kodlu veri setlerinde ise bu oran %21'e yakındır. Bu oranların ortaya çıkmasında projeler hazırlanırken takip edilen yazılım geliştirme metodolojileri etkili olmuş olabilir. Bununla beraber, yazılım süreç metriklerinin bu oranların elde edilmesindeki etkisi yadsınamaz. Geliştirici, zaman, çaba gibi metriklerin tekrar eden veri oranlarındaki etkisi araştırılması gereken bir başka yöndür.

6.3. Eğri Uydurma Analizleri

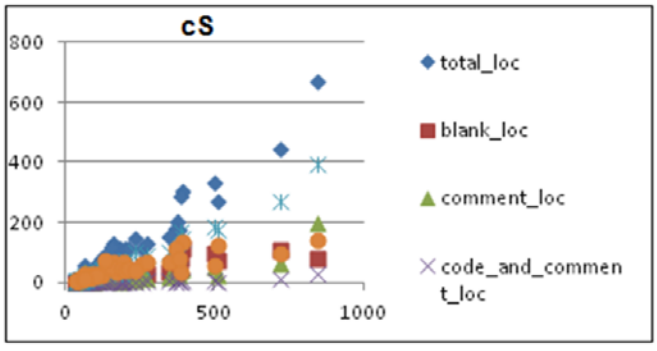
Şekil 6.1'de 63 örnekli ar3 veri setine ait cS-cCount eğri uydurma analizi sonuçları görülmektedir. Burada x ekseni cCount metriğini, y ekseni ise cS metriğini simgelemektedir. Şekil üzerinde gösterilen fonksiyonda x cCount, y ise cS dir. Bu iki metrik arasındaki doğrusal ilişki büyülüğü R^2 ile gösterilir. R^2 değeri 0.94 ile oldukça yüksektir.

Tablo 6.6. Elenen veri oranları

Veri Seti	tTest%	ki-kare%	Toplam örnek
mylyn	7	93	447
lucene	3	97	168
jdtCore	16	84	167
equinox	19	81	68
eclipsePde	11	89	317
ar1	41	59	22
ar3	40	60	10
ar4	48	52	25
ar5	90	10	11
ar6	53	47	34
cm1	28	72	35
jm1	92	8	1431
kc1	40	60	408
kc2	11	89	111
kc3	30	70	144
pc1	82	18	620
pc2	95	5	280
pc3	14	86	297
pc4	25	75	358
pc5	99	1	157



Şekil 6.1. Ar3 cCount-cS analizi



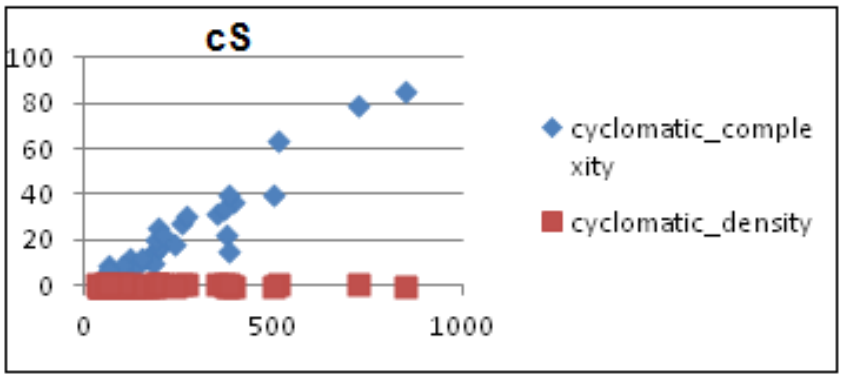
Şekil 6.2. Ar3 üzerinde cS ve diğer metrikler

Şekil 6.2'de cS ile diğer metrikler arasındaki dağılım gösterilmiştir. Burada cS gösterilmiştir. Burada cS *y* ile gösterilirse elde edilen denklemler eşitlikteki gibidir.

total_loc==> y = 0,6927x - 26,561,R² = 0,9427
code_and_comment_loc==> y = 0,4044x - 15,059, R² = 0,9469
unique_operands==> y = 0,1727x + 8,4236, R² = 0,7449

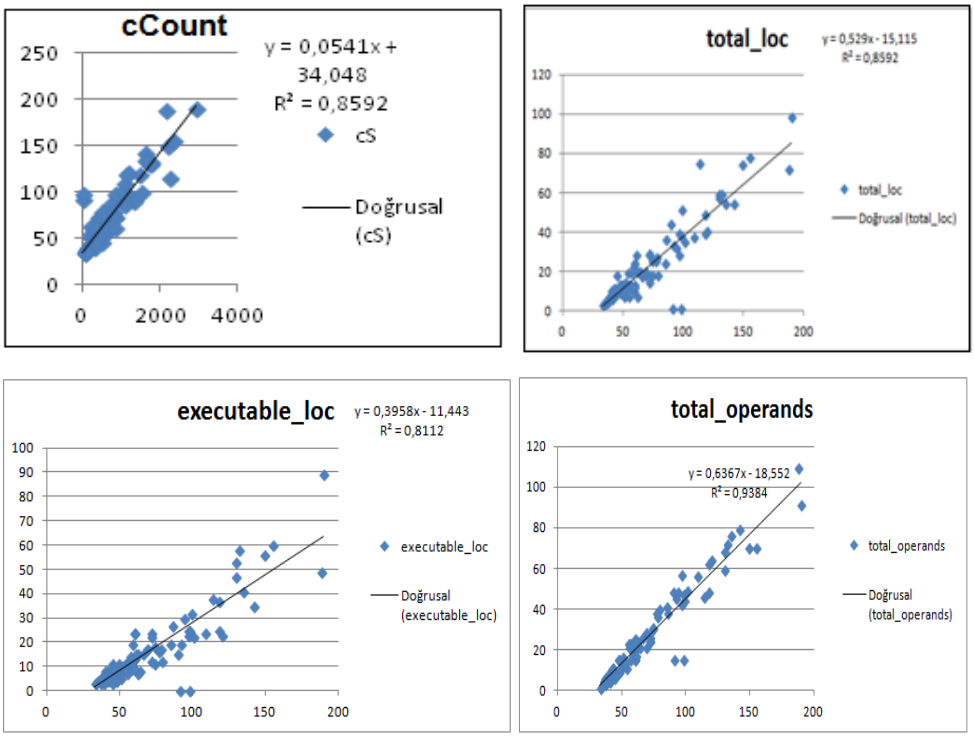
ar3 veri seti karmaşılık açısından değerlendirildiğinde yoğunluk parametresine göre daha doğrusal bir ilişki vardır denebilir. Şekil 6.3'te görüldüğü gibi yoğunluk metriği ile cS arasında doğrusal ilişki kuvveti oldukça düşüktür. Burada bu iki metrik ile cS metriği arasındaki doğrusal ilişki bulunmaya çalışılmışır.

Kullanılan deneysel verinin kolama biçimine ve diğer karakteristik özelliklerine bağlı olarak gerileme analizlerinde elde edilen eşitlikler değişebilir. Dolayısıyla daha genel veri setleri ile çalışmak doğru sonucun elde edilmesine yardımcı olmaktadır.



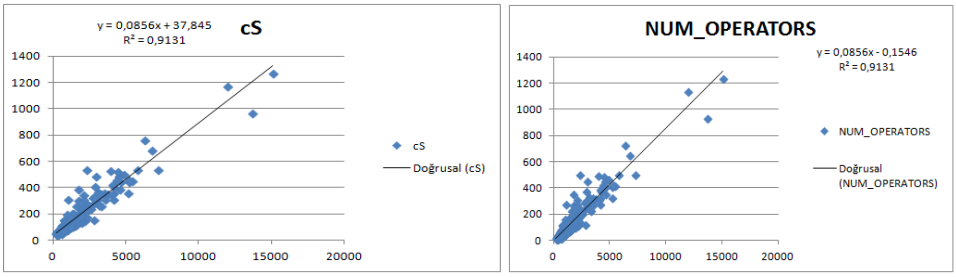
Şekil 6.3. cS-karmaşıklık ilişkileri

ar6 veri seti üzerindeki cCount-cS, cS-total_loc, cS-executable_loc, cS-total_operands parametre ilişkilerine bakılacak olursa total_operands ile cS arasında diğer metrik gruplarına göre daha doğrusal bir ilişkinin var olduğu 0.9 değerli R^2 ile görülebilmektedir. Bu veri setine ait regresyon grafikleri Şekil 6.4'te sunulmuştur.



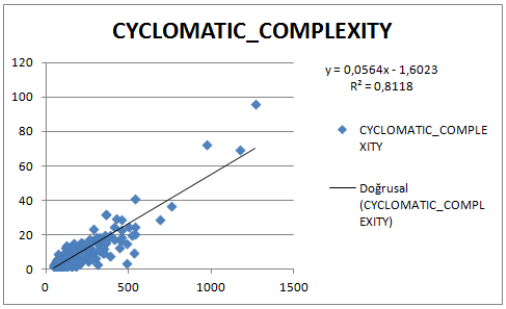
Şekil 6.4. ar6 veri setinde cS ve diğer metrikler

cm1 veri seti üzerinde yapılan regresyon analizlerinde Şekil 6.5'te görüldüğü üzere cS metriği ile diğer metrikler arasındaki ilişki 0.8'in üzerindedir. Dolayısıyla program karmaşıklığı ve cS arasında doğrusal bir ilişki vardır denebilir. Buradan cS metrik değerini azaltmak daha az karmaşık yani daha az hata yatkın yazılım demektir. R^2 sonuçlarına bakarak Şekil 6.5'teki metriklerin cS ile doğrusal bir ilişkisi olduğu söylenebilir.



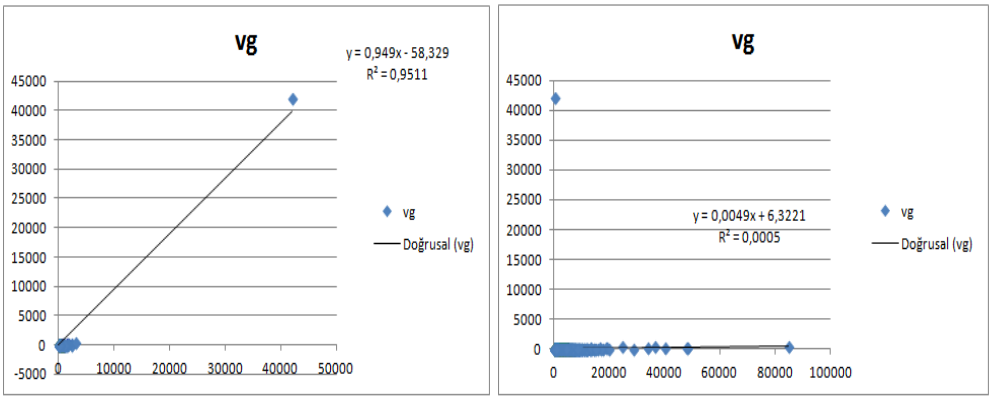
(a) cCount-cS (b) cS-num_operators

Şekil 6.5. cm1 veri setindeki analizler

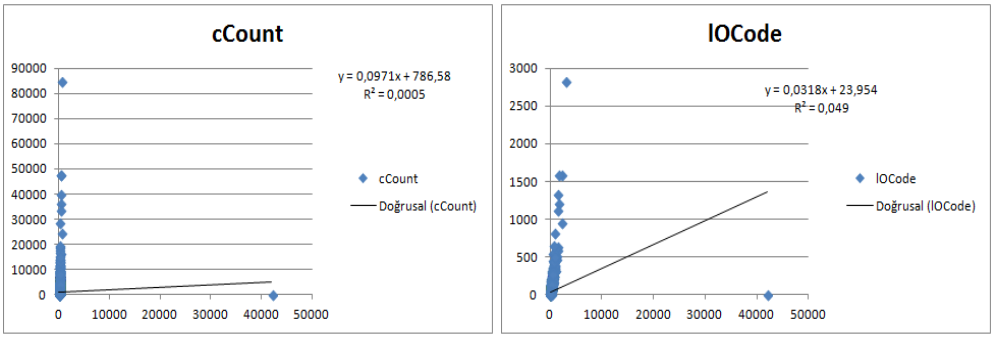


(c) cS-cyclomatic_complexity (d) cS-loc_executable

Şekil 6.5. (Devamı)



(a) cS-v(g) (b) cCount-v(g)



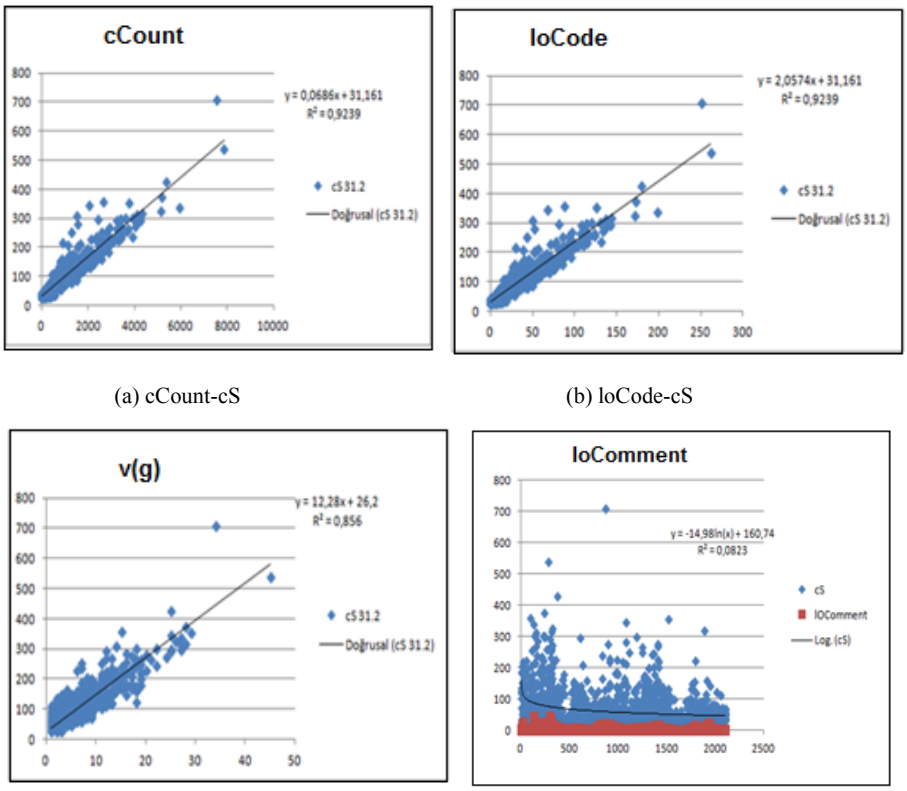
(c) v(g)-cCount (d) cS-loCode

Şekil 6.6. Jm1 veri setindeki analizler

Şekil 6.6'daki grafiklerden anlaşıldığı üzere sınıf büyüklüğü (cS) metriği ile karmaşıklık (complexity) arasında doğrusal bir ilişki vardır. Karmaşıklık karakter sayısı metriğinden tamamen bağımsız olup karakter sayısı ile karmaşıklık arasında az da olsa doğrusal bir ilişki vardır.

Karakter sayısı ile cS arasındaki lineer analizde elde edilen R^2 değeri diğer veri

setlerinde 0,8-0,91 aralığında iken kc1 veri setinde bu değer 0,92 olarak elde edilmiştir. Elde edilen denklemin daha güvenilir olduğunu buradan ifade edebiliriz. Benzer bir denklem Halstead loCode ile cS arasında da aynı R^2 değeri ile ölçülmüştür.

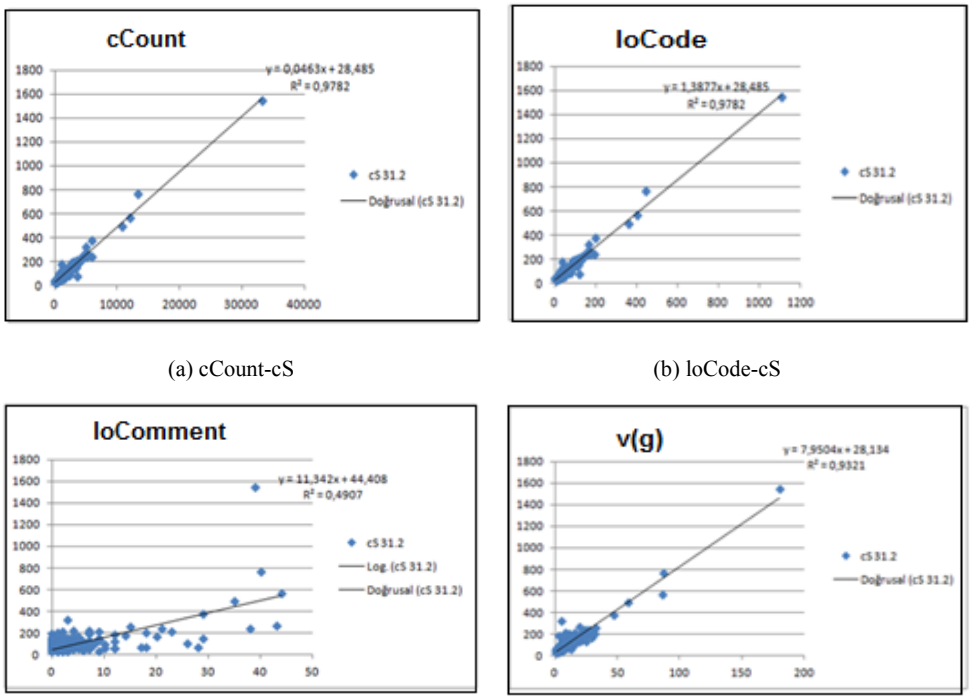


(a) cCount-cS (b) loCode-cS

(c) v(g)-cS (d) LoComment-cS

Şekil 6.7. Kc1 veri setindeki analizler

Şekil 6.7'de cm1 de elde edilen 0,81 R^2 değerinden daha iyi sonuç 0,85 ile kc1 veri setinde elde edilmiştir. Ancak burada bağımsız değişken katsayı bağımlılığı daha fazladır (Not:0,05-12,28). Yorum satırı sayısıyla sınıf büyüklüğü arasında da lineer olmayan bir ilişki söz konusudur. Dolayısıyla bu metriklerden çıkarım yapabilmek daha zordur. Diğer veri setlerindeki v(g)-cS doğrusallığını kc1 veri setinde de görmek mümkündür.

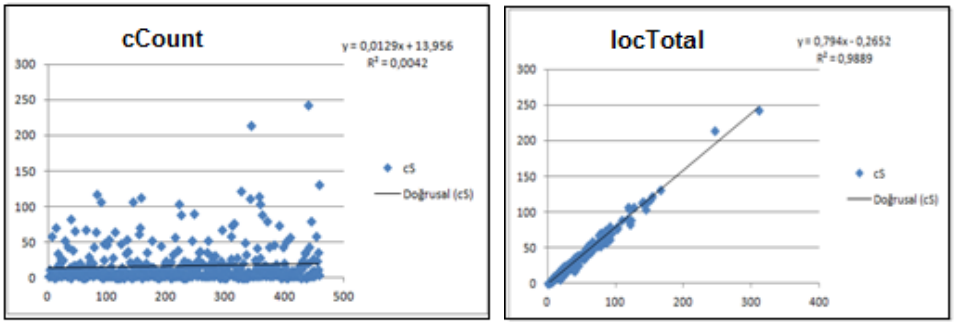


(a) cCount-cS (b) loCode-cS

(c) loComment-cS (d) v(g)-cS

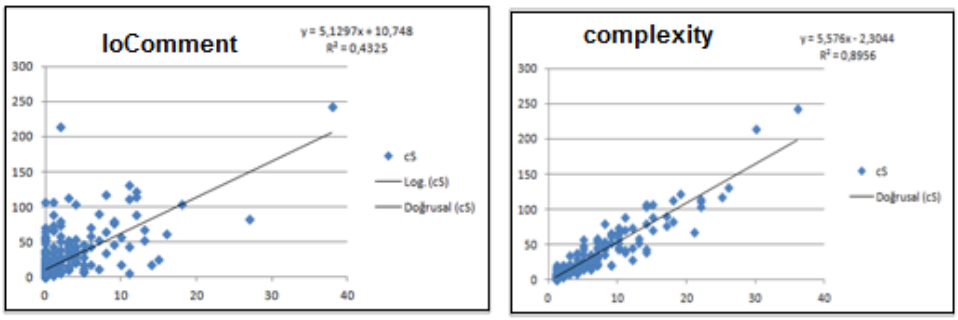
Şekil 6.8. Kc2 veri setindeki analizler

Kc2 veri setindeki analizlere bakıldığında ilişkilerin polinomal veya üstel olmadıklarını R^2 değerlerinin başındaki "0" değerlerinden anlayabiliyoruz. Şekil 6.8'de analiz detayları görülmektedir. Bu veri setinde de öncekilerde oldupu gibi yorum satırı metriği ile sınıf büyüklüğü arasında lineer ilişki seviyesi düşüktür. Diğer taraftan karmaşıklık ile sınıf büyüklüğü arasında lineer bir ilişki mevcuttur.



(a) cCount-Cs (b) LocTotal-cS

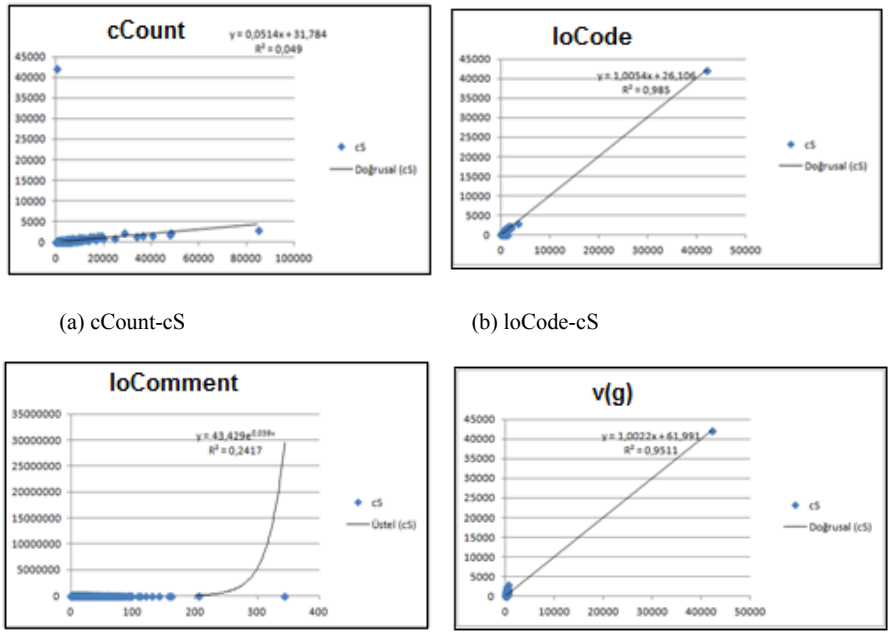
Şekil 6.9. Kc3 veri setindeki analizler



(c) LoComment-cS (d) Complexity-cS

Şekil 6.9. (Devamı)

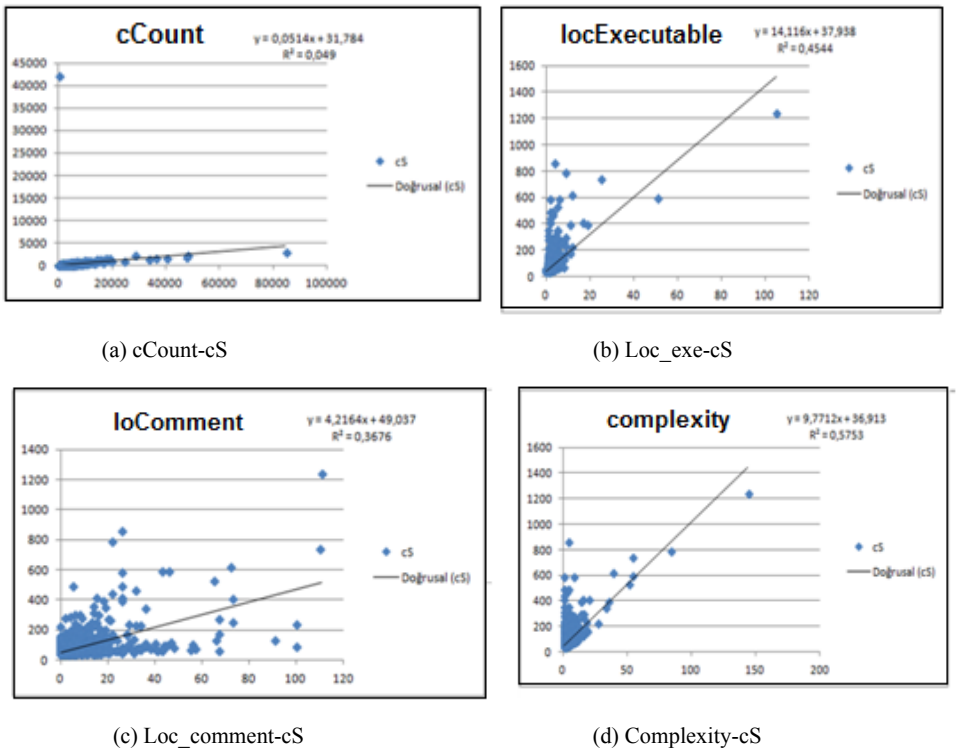
Kc3 veri setinde Şekil 6.9'daki analizlerde LOC ve türev metrikleri ile cS metriği arasında 0,9 ve üzeri bir doğrusal ilişki bulunmuştur. Bu da Loc ve cS'yi içeren yeni düşük seviyeli metriklerin araştırılması gerektiğini göstermektedir. pc1 veri setinde ise satır sayısı ile sınıf büyüklüğü arasıdanki ilişkiye benzer olarak karmaşıklık ile sınıf büyüklüğü arasında bir doğrusal ilişki 0.98 R^2 değeri ile elde edilmiştir. Bu veri setinin analiz detayları Şekil 6.10'da sunulmuştur.



(a) cCount-cS (b) loCode-cS

(c) loComment-cS (d) v(g)-cS

Şekil 6.10. Pc1 veri setindeki analizler

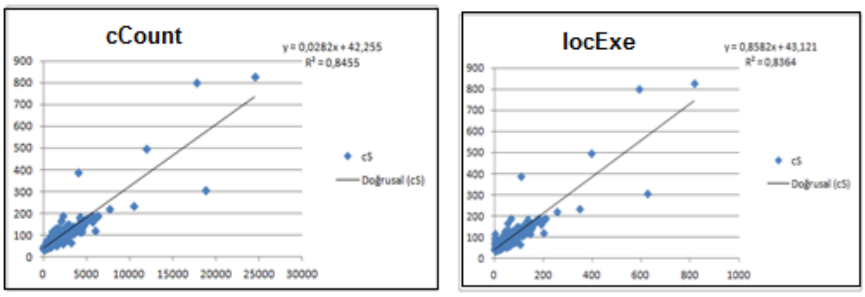


(a) cCount-cS (b) Loc_exe-cS

(c) Loc_comment-cS (d) Complexity-cS

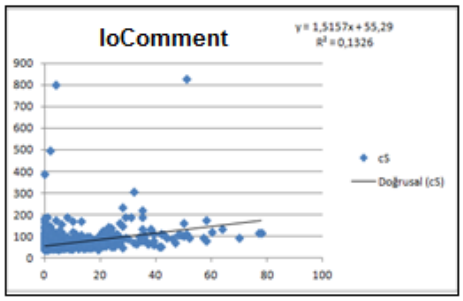
Şekil 6.11. Pc2 veri setindeki analizler

Pc2 veri setindeki analizler incelendiğinde sınıf büyüklüğü ile çalıştırılabilir satır sayısı arasındaki doğrusal ilişki düzeyinin zayıf, buna karşılık kamaşıklık ile sınıf büyüklüğü arasındaki doğrusallığın diğer veri setlerine göre 0.5 R^2 değeri ile en düşük seviyede bulunduğu söylenebilir. Pc3 veri setinde ise bu oran 0.05 düşmüştür. Şekil 6.11 ve 6.12'deki analiz sonuçlarından cS ile cCount metriklerinin birbirinden bağımsız oldukları görülmektedir.

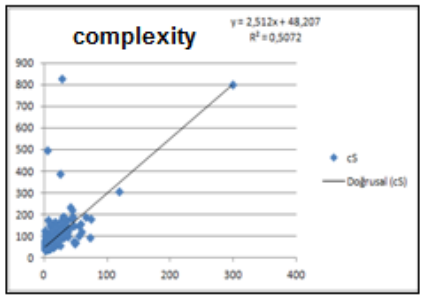


(a) cCount-cS (b) Loc_exe-cS

Şekil 6.12. Pc3 veri setindeki analizler

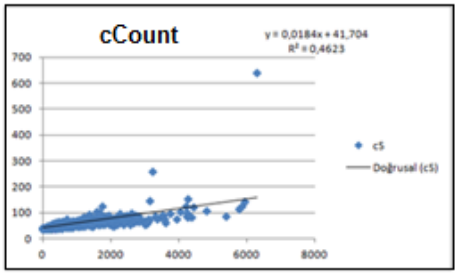


(c) Loc_comment-cS

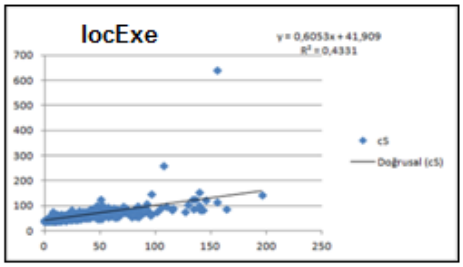


(d) complexity-cS

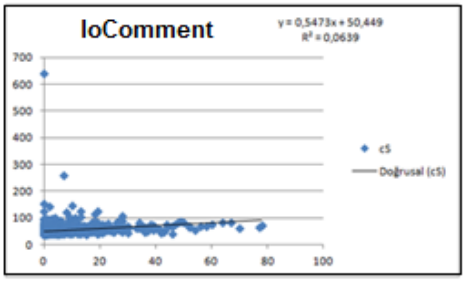
Şekil 6.12. (Devamı)



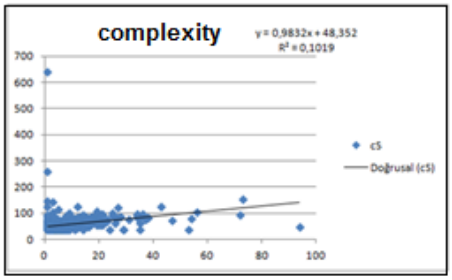
(a) cCount-cS



(b) Loc_exe-cS



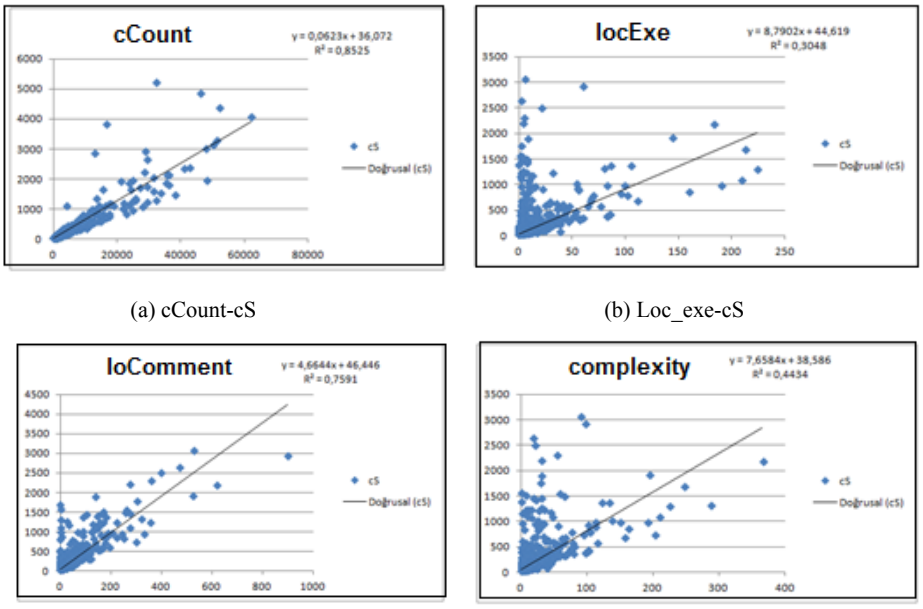
(c) Loc_comment-cS



(d) complexity-cS

Şekil 6.13. Pc4 veri setindeki analizler

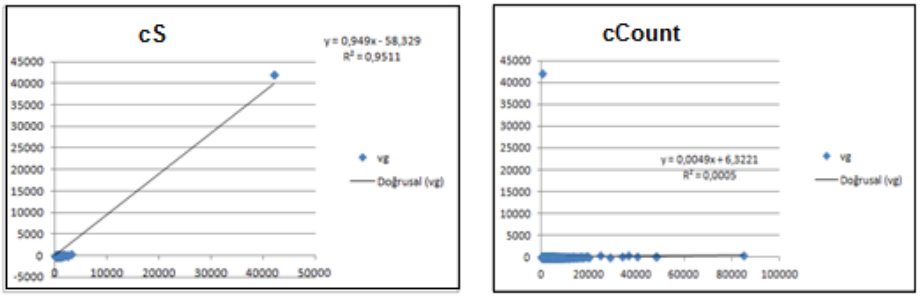
Pc4 veri setindeki regresyon analizleri diğer veri setleri ile karşılaştırıldığında tüm metrik gruplarında en düşük R^2 değerinin elde edildiği projedir. Ancak tüm metrik gruplarında yapılan analizlerde benzer R^2 sonuçlarının elde edilmesi bu proje ait bir özelliğin analiz sonuçlarını etkilediğini göstermektedir. Nitekim pc5 veri setinde bu hipotezin doğruluğunu kanıtlayan sonuçları görmek mümkündür. Ayrıca cS-cCount metriklerinin doğrusal ilişki seviyesi 0.8 ve üzeri değerlerde korunmuştur. Pc4 ve pc5 veri setlerine ait analiz sonuç detayları sırasıyla Şekil 6.13 ve 6.14'te görülmektedir.



(a) cCount-cS (b) Loc_exe-cS

(c) Loc_comment-cS (d) complexity-cS

Şekil 6.14. pc5 veri setindeki analizler



(a) cS-v(g) (b) cCount-v(g)

Şekil 6.15. Jm1 veri setindeki analiz sonuçları

Şekil 6.15'te görüldüğü gibi jm1 veri setinde karmaşıklık karakter sayısından bağımsızdır.

Sınıf büyüklüğü cS *y* fonksiyonu ile simgelenirse cCount *x* ile ilişkisi *y=xb+c* ile ifade edildiğinde bu değerlere ait değişim Tablo 6.7'deki gibidir. ar3-pc1 aralığında r değerimiz 0,8 üstündedir. Bu benzerlik C dilinde yazılan projelerde görülmektedir. Bağımsız değişken katsayının çok tutarlı olmadığını görmekteyiz. pc2-pc5 aralığında ise r değeri 0.1-0.5 aralığındadır.

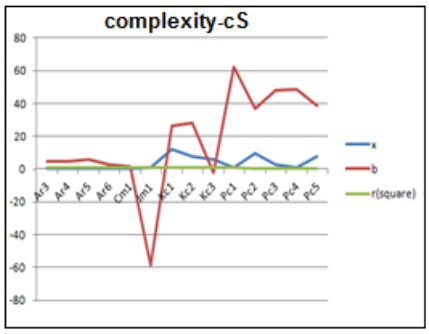
Tablo 6.7. Veri setleri cS-cCount regresyon analiz özeti

	jm1	kc1	kc2	kc3	pc1	pc2	pc3	pc4	pc5	ar1	ar6	cm1
x	0,004	0,068	0,046	0,012	0,051	0,081	0,028	0,018	0,062	0,060	0,054	0,085
b	6,322	31,16	28,48	13,95	31,78	41,91	42,25	41,70	36,07	30,12	34,04	37,84
r^2	0,005	0,923	0,923	0,004	0,049	0,833	0,845	0,462	0,852	0,945	0,859	0,913

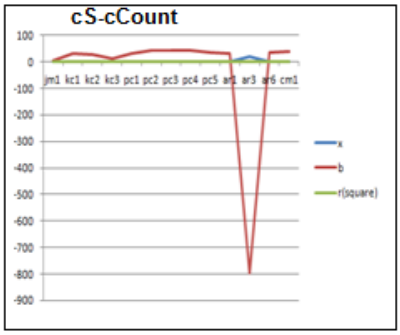
Tablo 6.8. Veri setleri complexity-cS regresyon analiz özeti

	ar3	ar4	ar5	ar6	cm1	jm1	kc1	kc2	kc3	pc1	pc2	pc3
x	0,105	0,105	0,125	0,096	0,056	0,949	12,28	7,95	5,576	1	9,771	2,512
b	4,43	4,43	5,974	2,666	1,602	58,32	26,2	28,13	-2,3	61,99	36,91	48,20
r^2	0,924	0,924	0,962	0,811	0,811	0,951	0,856	0,932	0,895	0,951	0,575	0,507

Bu veri setinin tipine göre değişmektedir. Grafikleri genel olarak değerlendirecek olursak şu sonuçları elde ederiz: Bağımsız değişken dağılımı ile sabit değer dağılımı tüm verilerde grafiklerdeki gibi benzerdir. Düşük örnekli verilerde (ar3-64 gibi) doğrusal analizler gerçekçi sonuçlar üretmemektedir. Kc1-Kc2 ve ar1, doğrusal analize en uygun değerleri üretmişlerdir (R^2). Tablo 6.7'den görüldüğü üzere cS ile complexity arasında doğrusal bir ilişki vardır. Karakter sayısı Cs arasında 0,8-0,9 arasında değişen bir doğrusal ilişki vardır. Tüm grafiklerde Loc ve türev metrikleri ile cS metriği arasında 0,9 ve üzeri bir doğrusal ilişki bulunmuştur. Bu da Loc ve cS'yi içeren yeni düşük seviyeli metriklerin araştırılması gerektiğini göstermektedir.



Şekil 6.16. Karmaşıklık-cS x-b-r grafiği



Şekil 6.17. cs-cCount x-b-r grafiği

Çoklu regresyon analizlerinde kc2 veri setinde 22 metrik değeri üzerinde yapılan analiz sonuçlarına göre ortalamalar ve diğer değerler iki grup veriyi ortaya çıkarmıştır. Tablo 6.9-6.17 aralığındaki tablolarda detaylar görülmektedir. 1. grup bağımsız değişkenle bağımlı değişken(cCount) arasında ilişki olduğunu diğer grup ise olmadığını göstermektedir. 10 metrik ile cCount arasında bir ilişki kurulamamış ancak 12 metrik

ile kurulabilmiştir. Bu sonuca P değerlerini kullanarak ulaşıyoruz. Sonucun güven değeri R^2 ile ifade edilmektedir. Bu veri setinde ve tüm veri setlerinde genel olarak şu sonuca ulaşabiliriz; tasarımla ilgili metriklerle cCount metriğinin herhangi bir ilişkisi yoktur. Bunu iv(g) ve essential_density gibi metriklerle görebiliriz. Ayrıca ilişkili metrik sayısı ilişkisiz metrik sayısından tüm veri setlerinde fazladır. Bu sonuçları özetlemek gerekirse sınıf büyüklüğü ile statik kod metrikleri arasında genel olarak doğrusal bir ilişki vardır. Ancak yoğunluk ile ilgili metriklerde durum tam aksidir.

Tablo 6.9. Pc5 çoklu regresyon sonuçları

Metrik No	**Y**	**P**	**R²**
1	cS	0	0,852513
2	LOC_BLANK	0	0,618699
3	BRANCH_COUNT	0	0,412721
4	CALL_PAIRS	0	0,293017
5	LOC_CODE_AND_COMMENT	0	0,156685
6	LOC_COMMENTS	0	0,669138
7	CONDITION_COUNT	0	0,43108
8	CYCLOMATIC_COMPLEXITY	0	0,410369
9	CYCLOMATIC_DENSITY	0,963151	6,21E-07
10	DECISION_COUNT	0	0,397909
11	DESIGN_COMPLEXITY	0	0,327821
13	DESIGN_COMPLEXITY	0	0,673514
14	DESIGN_DENSITY	0	0,251944
15	EDGE_COUNT	2,34E-36	0,009489
16	ESSENTIAL_COMPLEXITY	0	0,997886
17	ESSENTIAL_DENSITY	0,278541	6,83E-05
18	LOC_EXECUTABLE	0	0,282083
19	PARAMETER_COUNT	5,64E-43	0,011549
20	HALSTEAD_CONTENT	5,39E-16	0,004597
21	HALSTEAD_DIFFICULTY	1,22E-07	0,001718
22	HALSTEAD_EFFORT	0	0,714785
23	HALSTEAD_ERROR_EST	0	0,508158
24	HALSTEAD_LENGTH	0	0,862995
25	HALSTEAD_LEVEL	0,407082	0,000118
26	HALSTEAD_PROG_TIME	2,14E-49	0,024069
27	HALSTEAD_VOLUME	0,133064	0,000181

Tablo 6.9. (Devamı)

28	MAINTENANCE_SEVERITY	0	0,158695
29	MODIFIED_CONDITION_COUNT	0	0,438794
30	MULTIPLE_CONDITION_COUNT	0	0,387294
31	NODE_COUNT	0	0,710196
32	NORMALIZED_CYLOMATIC_COMPLEXITY	2,5E-298	0,692536
33	NUM_OPERANDS	0	0,852513
34	NUM_OPERATORS	0	0,852513
35	NUM_UNIQUE_OPERANDS	0	0,50209
36	NUM_UNIQUE_OPERATORS	0	0,166678
37	NUMBER_OF_LINES	0	0,942798
38	PERCENT_COMMENTS	0	0,113882
39	LOC_TOTAL	0	1

Tablo 6.10. Pc4 çoklu regresyon sonuçları

Metrik No	**Y**	**P**	**R^2**
1	cS	2,2E-198	0,462272
2	LOC_BLANK	5,7E-168	0,408021
3	BRANCH_COUNT	3,6E-139	0,351634
4	CALL_PAIRS	1,72E-86	0,234285
5	LOC_CODE_AND_COMMENT	6,4E-156	0,385037
6	LOC_COMMENTS	6E-99	0,263812
7	CONDITION_COUNT	8,27E-84	0,227776
8	CYCLOMATIC_COMPLEXITY	3E-140	0,353827
9	CYCLOMATIC_DENSITY	0,005383	0,061761
10	DECISION_COUNT	2,06E-75	0,207045
11	DECISION_DENSITY	1,66E-28	0,080815
12	DESIGN_COMPLEXITY	1,98E-90	0,243743
13	DESIGN_DENSITY		
14	EDGE_COUNT	1,6E-184	0,438173
15	ESSENTIAL_COMPLEXITY	2,43E-74	0,204359
16	ESSENTIAL_DENSITY	0,056801	0,00249
17	LOC_EXECUTABLE	0	0,965121
18	PARAMETER_COUNT	0,144763	0,00146
19	HALSTEAD_CONTENT	1,2E-17	0,122365
20	HALSTEAD_DIFFICULTY	1,72E-06	0,022789
21	HALSTEAD_EFFORT	8,65E-92	0,888008
22	HALSTEAD_ERROR_EST	3,62E-26	0,527086

Tablo 6.10. (Devamı)

23	HALSTEAD_LENGTH	0	0,69294
24	HALSTEAD_LEVEL		
25	HALSTEAD_PROG_TIME	0,366219	0,002778
26	HALSTEAD_VOLUME	0,427302	0,002454
28	MODIFIED_CONDITION_COUNT	7,87E-89	0,239918
29	MULTIPLE_CONDITION_COUNT	1,85E-81	0,222029
30	NODE_COUNT	1,4E-188	0,445316
31	NORMALIZED_CYLOMATIC_COMPLEXITY	0,368828	0,014981
32	NUM_OPERANDS	0	0,670107
33	NUM_OPERATORS	0	0,654578
34	NUM_UNIQUE_OPERANDS	0	0,654578
35	NUM_UNIQUE_OPERATORS	9,7E-227	0,508355
36	NUMBER_OF_LINES	7,3E-210	0,481426
37	PERCENT_COMMENTS	4,07E-48	0,198406
38	LOC_TOTAL	0	1

Tablo 6.11. Pc3 çoklu regresyon sonuçları

Metrik No	**Y**	**P**	**R^2**
1	cS	0	0,845513
2	LOC_BLANK	1,27E-81	0,209154
3	BRANCH_COUNT	2,5E-243	0,508917
4	CALL_PAIRS	1,69E-68	0,178018
5	LOC_CODE_AND_COMMENT	1,68E-27	0,072868
6	LOC_COMMENTS	1,19E-68	0,178392
7	CONDITION_COUNT	2E-221	0,476162
8	CYCLOMATIC_COMPLEXITY	5,9E-249	0,516992
9	CYCLOMATIC_DENSITY	1,5E-49	0,985584
10	DECISION_COUNT	2,1E-207	0,454059
11	DECISION_DENSITY	1,65E-18	0,048189
12	DESIGN_COMPLEXITY	3,4E-118	0,289898
14	EDGE_COUNT	1,5E-253	0,523504
15	ESSENTIAL_COMPLEXITY	2,33E-81	0,208542
16	ESSENTIAL_DENSITY	0,648634	0,000164
17	LOC_EXECUTABLE	0	0,993837
18	PARAMETER_COUNT	6,16E-06	0,013012
19	HALSTEAD_CONTENT	4,38E-13	0,105095
20	HALSTEAD_DIFFICULTY	3,42E-08	0,027614

Tablo 6.11. (Devamı)

21	HALSTEAD_EFFORT	2,68E-44	0,82027
22	HALSTEAD_ERROR_EST	2,11E-15	0,404358
23	HALSTEAD_LENGTH	0	0,751736
24	HALSTEAD_LEVEL	0,940078	0,000116
25	HALSTEAD_PROG_TIME	0,671549	0,000701
26	HALSTEAD_VOLUME	0,045384	0,020693
28	MODIFIED_CONDITION_COUNT	1,1E-231	0,491756
29	MULTIPLE_CONDITION_COUNT	2,1E-220	0,474584
30	NODE_COUNT	3,1E-251	0,520224
31	NORMALIZED_CYLOMATIC_COMPLEXITY	2,4E-219	1
32	NUM_OPERANDS	0	0,795836
33	NUM_OPERATORS	0	0,706754
34	NUM_UNIQUE_OPERANDS	0	0,706754
35	NUM_UNIQUE_OPERATORS	4,2E-114	0,281278
36	NUMBER_OF_LINES	0	0,910536
37	PERCENT_COMMENTS	5,11E-30	0,110779

Tablo 6.12. Pc2 çoklu regresyon sonuçları

Metrik No	**Y**	**P**	**R^2**
1	cS	0	0,833313
2	BRANCH_COUNT	0	0,820704
3	CALL_PAIRS	0	0,594857
4	LOC_CODE_AND_COMMENT	0	0,995026
5	LOC_COMMENTS	0	0,37967
6	CONDITION_COUNT	0	0,728003
7	CYCLOMATIC_COMPLEXITY	0	0,820839
8	CYCLOMATIC_DENSITY	7,17E-19	0,048405
9	DECISION_COUNT	0	0,723993
10	DECISION_DENSITY	8,77E-93	0,072001
11	DESIGN_COMPLEXITY	0	0,771294
13	EDGE_COUNT	0	0,923027
14	ESSENTIAL_COMPLEXITY	0	0,567208
15	ESSENTIAL_DENSITY	2,03E-11	0,008013
16	LOC_EXECUTABLE	0	0,760892
17	PARAMETER_COUNT	3,34E-15	0,011046
18	HALSTEAD_CONTENT	6,01E-62	0,059723
19	HALSTEAD_DIFFICULTY	4,64E-06	0,004208

Tablo 6.12. (Devamı)

20	HALSTEAD_EFFORT	2,35E-29	0,058436
21	HALSTEAD_ERROR_EST	5,9E-215	0,422677
22	HALSTEAD_LENGTH	0	0,834891
23	HALSTEAD_LEVEL	1,06E-07	0,037846
24	HALSTEAD_PROG_TIME	6,43E-32	0,047967
25	HALSTEAD_VOLUME	0,000209	0,005158
27	MODIFIED_CONDITION_COUNT	0	0,721078
28	MULTIPLE_CONDITION_COUNT	0	0,727762
29	NODE_COUNT	0	0,905892
30	NORMALIZED_CYLOMATIC_COMPLEXITY	3,14E-08	0,029816
31	NUM_OPERANDS	0	0,8178
32	NUM_OPERATORS	0	0,833313
33	NUM_UNIQUE_OPERANDS	0	0,743936
34	NUM_UNIQUE_OPERATORS	0	0,743936
35	NUMBER_OF_LINES	0	0,942611
36	PERCENT_COMMENTS	0	0,420967

Tablo 6.13. Pc1 çoklu regresyon sonuçları

Metrik No	**Y Label**	**P**	**R²**
1	cS	5,5356E-121	0,049026
2	loc	3,03521E-70	0,028436
3	vg	0,022292818	0,00048
4	evg	0,697189125	1,39E-05
5	ivg	0,155055135	0,000186
6	n	0	0,244646
7	v	0,816937589	2,14E-05
8	l	0,955369869	2,27E-06
9	d	1,21628E-60	0,03403
10	i	1,0562E-156	0,160432
11	e	0	0,644035
12	b	0	0,499617
13	t	4,41249E-09	0,011749
14	lOCode	0	1
15	lOComment	0	0,305943
16	lOBlank	0	0,651765
17	locCodeAndComment	1,5854E-166	0,06715
18	uniq_Op	0,193007495	0,000156

Tablo 6.13. (Devamı)

19	uniq_Opnd	5,32531E-08	0,002716
20	total_Op	2,7026E-272	0,107942
21	total_Opnd	5,5356E-121	0,049026
22	branchCount	2,21075E-05	0,001653

Tablo 6.14. Kc1 çoklu regresyon sonuçları

Metrik No	**Y**	**P**	**R²**
1	cS	5,5356E-121	0,049026
2	loc	3,03521E-70	0,028436
3	vg	0,022292818	0,00048
4	evg	0,697189125	1,39E-05
5	ivg	0,155055135	0,000186
6	n	0	0,244646
7	v	0,816937589	2,14E-05
8	l	0,955369869	2,27E-06
9	d	1,21628E-60	0,03403
10	i	1,0562E-156	0,160432
11	e	0	0,644035
12	b	0	0,499617
13	t	4,41249E-09	0,011749
14	lOCode	0	1
15	lOComment	0	0,305943
16	lOBlank	0	0,651765
17	locCodeAndComment	1,5854E-166	0,06715
18	uniq_Op	0,193007495	0,000156
19	uniq_Opnd	5,32531E-08	0,002716
20	total_Op	2,7026E-272	0,107942
21	total_Opnd	5,5356E-121	0,049026
22	branchCount	2,21075E-05	0,001653

Tablo 6.15. Kc2 çoklu regresyon sonuçları

Metrik No	**Y**	**P**	**R²**
1	cS	0	0,978253563
2	loc	0,576175	0,000601281
3	vg	0,787974	0,000139221
4	evg	0,746036	0,000201889

Tablo 6.15. (Devamı)

5	ivg	0,761631	0,000177106
6	n	0,013663	0,011637663
7	v	0,008668	0,013175193
8	l	0,561584	0,000648432
9	d	5,86E-08	0,055036566
10	i	9,75E-08	0,053240726
11	e	4,2E-235	0,872926766
12	b	1,01E-59	0,400293033
13	t	3,87E-12	0,088590475
14	lOCode	0	1
15	lOComment	3,73E-80	0,49935369
16	lOBlank	3,2E-199	0,825417162
17	locCodeAndComment	1,75E-52	0,360727212
18	uniq_Op	0,726298	0,000235849
19	uniq_Opnd	0,890011	3,68123E-05
20	total_Op	0,175319	0,003529591
21	total_Opnd	0,47908	0,00096384
22	branchCount	0,890758	3,63102E-05

Tablo 6.16. jm1 çoklu regresyon sonuçları

Örnek	**Y**	**P**	**R²**
1	cS	5,5356E-121	0,049026
2	loc	3,03521E-70	0,028436
3	vg	0,022292818	0,00048
4	evg	0,697189125	1,39E-05
5	ivg	0,155055135	0,000186
6	n	0	0,244646
7	v	0,816937589	2,14E-05
8	l	0,955369869	2,27E-06
9	d	1,21628E-60	0,03403
10	i	1,0562E-156	0,160432
11	e	0	0,644035
12	b	0	0,499617
13	t	4,41249E-09	0,011749
14	lOCode	0	1
15	lOComment	0	0,305943
16	lOBlank	0	0,651765
17	locCodeAndComment	1,5854E-166	0,06715
18	uniq_Op	0,193007495	0,000156
19	uniq_Opnd	5,32531E-08	0,002716

Tablo 6.16. (Devamı)

20	total_Op	2,7026E-272	0,107942
21	total_Opnd	5,5356E-121	0,049026
22	branchCount	2,21075E-05	0,001653

Tablo 6.15'de cCount metriği ile diğer metrikler arasındaki çoklu regresyon analiz sonuçlarına ait ilişkili ve ilişkisiz metrik sayıları görülmektedir. İlişkili metrik sayısı ile ilişkisiz metrik sayısı benzerdir. Tasarımla ilgili metriklerin karakter sayısı ile ilişkisiz olduğu anlaşılmaktadır.

Tablo 6.17. cCount çoklu regresyon sonuçları

Proje ismi	**İlişkili metrik sayısı**	**İlişkisiz metrik sayısı**
kc2	12	10
jm1	17	5
kc1	12	9
kc3	33	5
pc1	22	3
pc2	33	3
pc3	33	3
pc4	33	5
pc5	35	4

6.4. Deney Sonuçları

6.4.1 Düşük seviyeli metriklerin etkisi

Deneyin birinci aşamasında düşük seviyeli metrik türetiminde tahmin başarısının değişimi gözlenmiştir. İki adet düşük seviyeli metrik cCount ve cS eklendikten sonraki değişimler AUC parametresi açısından Tablo 6.18 ve Tablo 6.19'da gösterilmiştir.

Düşük seviyeli metrikler tüm veri setlerine eklendikten sonra görülmektedir ki jm1-pc3 veri seti aralığının tümünde 0,1 ile 0,4 arasında değişen oranlarda AUC değerleri yükselmiştir.

Tablo 6.18. Düşük seviyeli metrikler eklendikten sonraki AUC değerleri

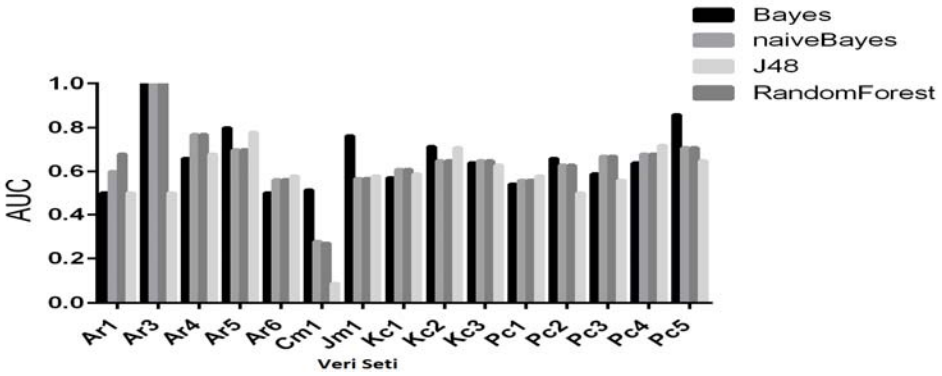
	Bayes	**naiveBayes**	**Rastgele Orman**	**J48**
Ar1	0,5 -hata:0,175	0,6-hata:0,176	0,68-hata:0,176	0,5 -hata:0,175
Ar3	1-hata:0	1-hata:0	1-hata:0	0,5-hata:0,3
Ar4	0,66-hata:0,122	0,77-hata:0,111	0,77-hata:0,11	0,68-hata:0,12
Ar5	0,8-hata:0,138	0,7-hata:0,158	0,7-hata:0,15	0,78-hata:0,14
Ar6	0,5-hata:0,131	0,563-hata:0,133	0,563-hata:0,133	0,58-hata:0,13
Cm1	0,514-hata:0,09	0,279-hata:0,09	0,27-hata:0,094	0,09-hata:0,06
Jm1	0,764-hata:0,01	0,569-hata:0,01	0,569-hata:0,01	0,58-hata:0,012
Kc1	0,571-hata:0,02	0,61-hata:0,02	0,61-hata:0,02	0,59-hata:0,02
Kc2	0,715-hata:0,051	0,65-hata:0,05	0,65-hata:0,05	0,71-hata:0,05
Kc3	0,64-hata:0,08	0,65-hata:0,08	0,65-hata:0,08	0,63-hata:0,08
Pc1	0,54-hata:0,012	0,56-hata:0,01	0,56-hata:0,01	0,58-hata:0,012
Pc2	0,66-hata:0,094	0,63-hata:0,09	0,63-hata:0,09	0,5-hata:0,09
Pc3	0,59-hata:0,0376	0,67-hata:0,03	0,67-hata:0,03	0,56-hata:0,03
Pc4	0,64-hata:0,04	0,68-hata:0,03	0,68-hata:0,03	0,72-hata:0,03
Pc5	0,86-hata:0,01	0,71-hata:0,02	0,71-hata:0,02	0,65-hata:0,02

Tablo 6.19. Düşük seviyeli metrikler eklenmeden önceki AUC değerleri

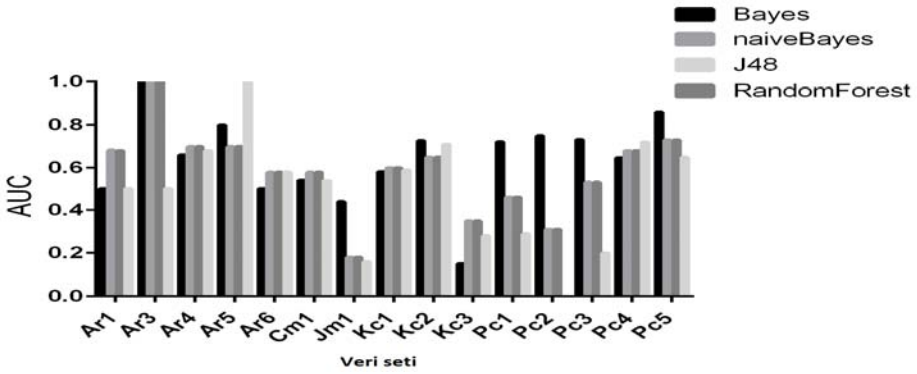
	Bayes	**naiveBayes**	**Rastgele Orman**	**J48**
Ar1	0,5 -hata:0,175	0,683-hata:0,176	0,68-hata:0,017	0,5 -hata:0,175
Ar3	1-hata:0	1-hata:0	1-hata:0	0,5-hata:0,3
Ar4	0,66-hata:0,122	0,7-hata:0,11	0,7-hata:0,11	0,68-hata:0,12
Ar5	0,8-hata:0,138	0,7-hata:0,15	0,7-hata:0,15	078-hata:0,14
Ar6	0,5-hata:0,131	0,58-hata:0,133	0,58-hata:0,133	0,58-hata:0,13
Cm1	0,542-hata:0,075	0,58-hata:0,07	0,58-hata:0,07	0,54-hata:0,07
Jm1	0,44-hata:0,013	0,18-hata:0,011	0,18-hata:0,011	0,16-hata:0,01
Kc1	0,582-hata:0,002	0,60-hata:0,02	0,60-hata:0,02	0,59-hata:0,02
Kc2	0,727-hata:0,05	0,65-hata:0,05	0,65-hata:0,05	0,71-hata:0,05
Kc3	0,15-hata:0,07	0,35-hata:0,09	0,35-hata:0,09	0,28-hata:0,08
Pc1	0,722-hata:0,05	0,46-hata:0,06	0,46-hata:0,06	0,29-hata:0,06
Pc2	0,75-hata:0,08	0,31-hata:0,13	0,31-hata:0,13	0-0
Pc3	0,732-hata:0,03	0,53-hata:0,04	0,53-hata:0,04	0,2-hata:0,03
Pc4	0,647-hata:0,04	0,68-hata:0,03	0,68-hata:0,03	0,72-hata:0,03
Pc5	0,86-hata:0,01	0,73-hata:0,02	0,73-hata:0,02	0,65-hata:0,02

Bu değişim düşük seviyeli metrik kullanımının tahmin başarına etkisinin olumlu

olduğunu göstermektedir. Diğer veri seti gruplarında sadece iki veri seti grubunda(ar6,cm1,) düşüş gözlenmiş kalan veri setlerinde ise başarı korunmuştur.



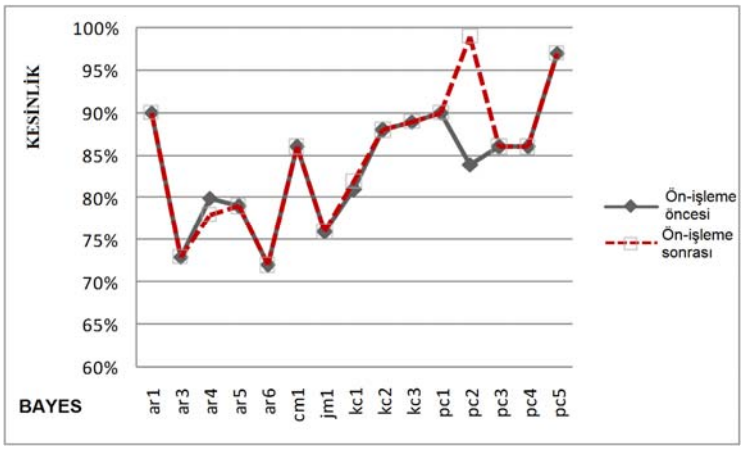
Şekil 6.18. Metrik türetimi sonrası AUC değerleri



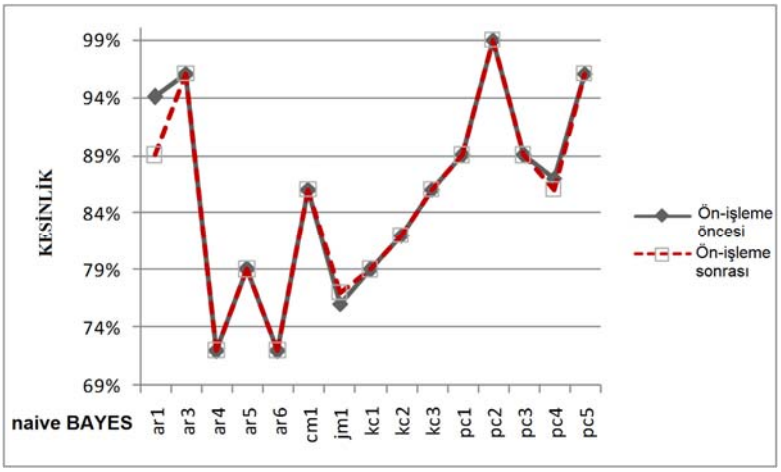
Şekil 6.19. Metrik türetimi öncesi AUC değerleri

Şekil 6.18 ve Şekil 6.19'da metrik türetimi sonrası ve metrik türetimi öncesi elde edilen AUC değerlerine ait grafikler görülmektedir. Bu grafiklere bakılarak metrik türetiminin AUC performans parametresini özellikle J48 ve Rastgele Orman sınıflandırıcılarında arttırtığı gözlenmiştir.

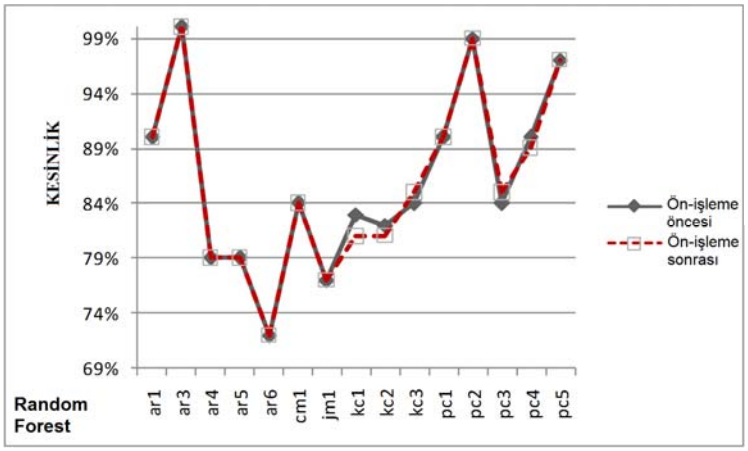
Şekil 6.20-6.23 aralığındaki şekillerde metrik türetiminin endüstriyel veri setlerindeki etkisi Kesinlik performans parametresi açısından gözlenmiştir. Ön-işleme öncesi değerler kırmızı-kesiksiz, ön-işleme sonrası değerler ise kırmızı-kesikli çizgilerle gösterilmiştir. Böyle grafiklerde iki çizginin çakışması beklenir. Bu noktadan bakıldığında Naive Bayes daha iyi sonuçlar üretmiştir denebilir.



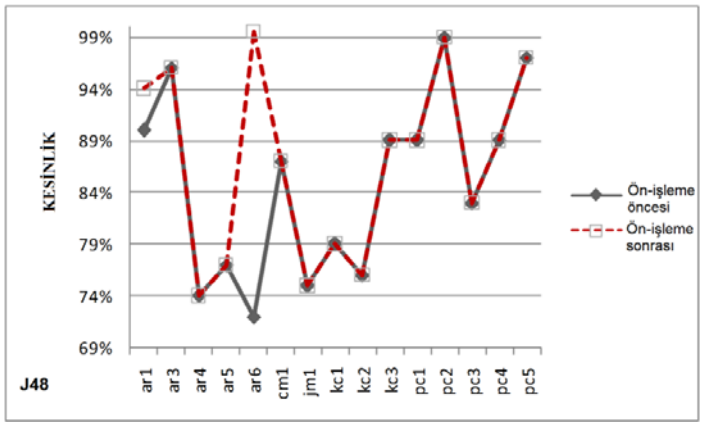
Şekil 6.20. Metrik türetiminin Bayes sınıflandırıcı üzerindeki etkisi



Şekil 6.21. Metrik türetiminin Naive Bayes sınıflandırıcı üzerindeki etkisi



Şekil 6.22. Metrik türetiminin Rastgele Orman sınıflandırıcısı üzerindeki etkisi



Şekil 6.23. Metrik türetiminin J48 sınıflandırıcı üzerindeki etkisi

Bayes sınıflandırıcısı pc1, pc2 ve pc3 veri setlerinde başarıyı arttırmış fakat genel bir tutarlılık göstermemiştir. Performanstaki en yüksek iyileşme %4 oranla Rastgele Orman algoritması ve pc2 veri setinde elde edilmiştir.

6.4.2. Düşük seviyeli metrik t-test

Bu testi yapmamızdaki amaç tüm veri setleri üzerinde karakter sayısı ile sınıf büyüklüğü arasında varyans açısından önemli bir fark olduğunu göstermektir. Böylece tüm veri setlerinde kullanılan bu iki düşük seviyeli metrik birbirinden bağımsızdır denebilir. Yani metrik değerlerinin herhangi birindeki artış diğerini etkilememektedir. Yazılım tasarımı sırasında bu nokta dikkate alınmalıdır. Tablo 6.20, Tablo 6.21, Tablo 6.22 ve Tablo 6.23 t-test sonuç tablolarından anlaşıldığı üzere cS-cCount metrikleri birbirinden bağımsız metriklerdir. Tüm veri setlerine ait *p* değerleri Tablo 6.22'de özetlenmiştir.

Tablo 6.20. ar3 veri seti cs-cCount t-Test

	cS	cCount
Ortalama	611,6529	66,87603306
Varyans	345785,6	1320,859504
Gözlem	121	121
Öngörülen Ortalama Farkı	0	
df	121	

Tablo 6.20. ar3 veri seti cs-cCount t-Test (Devamı)

t Stat	10,17138
P(T<=t) tek-uçlu	3,07E-18
t Kritik tek-uçlu	1,657544
P(T<=t) iki-uçlu	6,15E-18
t Kritik iki-uçlu	1,979764

Tablo 6.21. ar4 veri seti cs-cCount t-Test

	cS	cCount
Ortalama	611,6529	66,87603306
Varyans	345785,6	1320,859504
Gözlem	121	121
Öngörülen Ortalama Farkı	0	
df	121	
t Stat	10,17138	
P(T<=t) tek-uçlu	3,07E-18	
t Kritik tek-uçlu	1,657544	
P(T<=t) iki-uçlu	6,15E-18	
t Kritik iki-uçlu	1,979764	

Tablo 6.22. ar5 veri seti cs-cCount t-Test

	cS	cCount
Ortalama	2678,095238	167,2222222
Varyans	12263725,35	26773,85305
Gözlem	63	63
Öngörülen Ortalama Farkı	0	
df	62	
t Stat	5,684735501	
P(T<=t) tek-uçlu	1,88505E-07	
t Kritik tek-uçlu	1,669804163	
P(T<=t) iki-uçlu	3,7701E-07	
t Kritik iki-uçlu	1,998971498	

Tablo 6.23. ar6 veri seti cs-cCount t-Test

	cS	cCount
Ortalama	2276,667	157,4722
Varyans	7311023	20270,94
Gözlem	36	36
Öngörülen Ortalama Farkı	0	
df	35	

Tablo 6.23. (Devamı)

t Stat	4,69604
P(T<=t) tek-uçlu	2E-05
t Kritik tek-uçlu	1,689572
P(T<=t) iki-uçlu	4E-05
t Kritik iki-uçlu	2,030108

Tablo 6.24. Özet cs-cCount t-Test

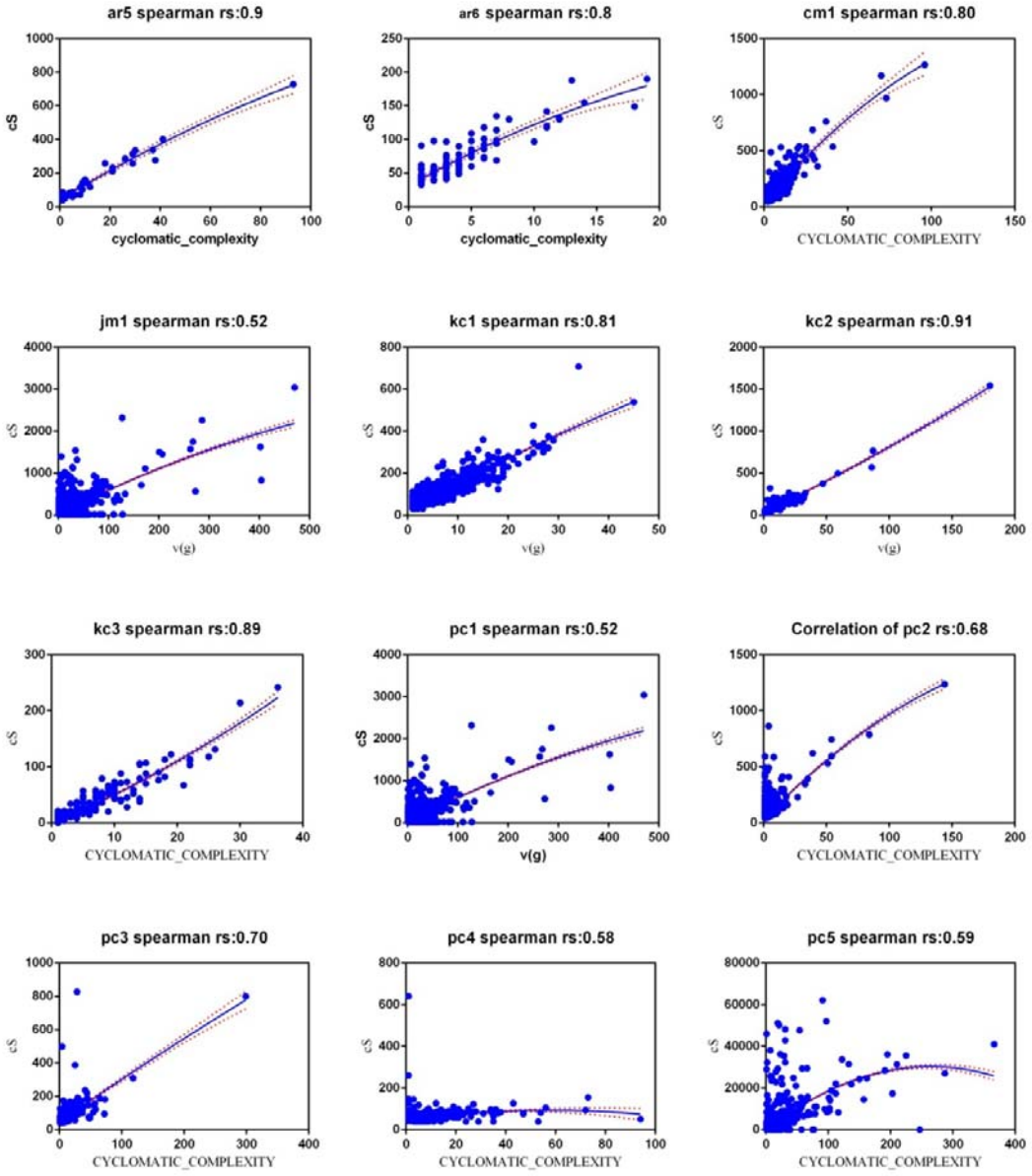
Veri Seti	**cCount-cS T-test (P)**
Ar1	6,15E-18
Ar3	3,7701E-07
Ar4	3,7701E-07
Ar5	4E-05
Ar6	4,14E-15
Cm1	9,34602E-33
Jm1	0
Kc1	5E-110
Kc2	2,91E-18
Kc3	8,37E-16
Pc1	0
Pc2	1,1E-51
Pc3	3,47E-81
Pc4	2E-142
Pc5	4,63E-57

6.5. Spearman Analizleri

Spearman monotik ilişki araştırmaları türetilen metriklerde NASA MDP veri setlerinde yürütülmüştür. V(g)-Complexity metrikleri ile cS metriği arasındaki lineer ilişki düzeyinin araştırılmasının temel nedeni cCount metriğinin doğal olarak diğer metriklerle belirli bir düzeyde bağlantısının olması ancak cS metriğinde doğrudan bir ilişkinin olmamasıdır. Karmaşıklık hata yatkınlığını arttırdığına göre bu karmaşıklıkla sınıf büyüklüğü arasındaki lineer ilişki ölçümü açıklayıcı olacaktır. Endüstriyel veri setlerine ait analiz sonuçları Şekil 6.24'te görülmektedir. Sonuçlara göre cs ile v(g) arasında monotonik artan bir ilişki mevcuttur. Sınıf tasarımına bağlı olarak karmaşıklık değişmektedir.

6.6. Tekrar Eden Veri Oranları

Açık kaynak kodlu projelerde tekrar eden veri oranları 20-24 arasında beş projede değişmektedir. Örnek sayısına bağlı olmaksızın elde edilen oranlardır. Bu da proje büyüklüklerine bağlı olarak tekrar eden veri oranlarının değişmediğini, ancak proje tipine bağlı olduğunu göstermektedir.



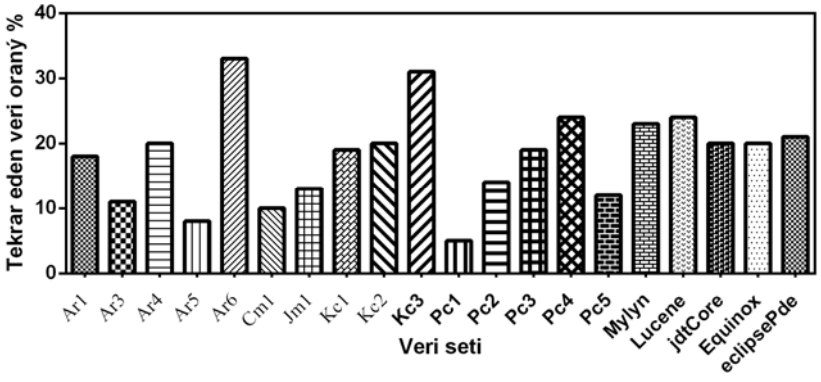
Şekil 6.24. Spearman analiz sonuçları

Nasa ve SOFTLAB veri setlerinde C ve Java dillerinde yazılan projelerde tekrar eden veri oranları benzerlik göstermezken, **C++** dilinde yazılan Kc1 ve kc2 projelerinde aynı oranda tekrar eden veri bulunmuştur. Bu projelerde örnek sayısı farkı 1500 dür. Bu da proje büyüklüğüne değil, geliştirici, geliştirme dili gibi diğer etkenlerin tekrar eden veri oranında etkili olduğunu göstermektedir.

Nasa veri setlerindeki sonuçlar ile (%17 tekrar eden veri) açık kaynak kodlu projelerdeki tekrar oranı (%21) birbirine yakındır. Tablo 6.25'te tekrar eden veri oranları ve Şekil 6.25'te bu oranların karşılaştırma grafiği sunulmuştur. Yazılım projelerinde izlenen metodolojilerin benzer olması bu yakınlığı sağlamış olabilir. Bununla beraber tekrar eden veri oranlarında süreç metriklerinin belirli düzeyde etkisi vardır. Dolayısıyla metrik tabloları hazırlanırken süreç metriklerinin oluşturulması, tekrar eden veri oranına geliştirici, zaman, çaba gibi etmenlerin etkisinin araştırılması bakımından önemlidir.

Tablo 6.25.Tekrar eden veri oranları

Veri Seti	**Yüzde (%)**
Ar1	18
Ar3	11
Ar4	20
Ar5	8
Ar6	33
Cm1	10
Jm1	13
Kc1	19
Kc2	20
Kc3	31
Pc1	5
Pc2	14
Pc3	19
Pc4	24
Pc5	12
mylyn	23
lucene	24
jdtCore	20
equinox	20
eclipsePde	21



Şekil 6.25. Tekrar eden veri oranları

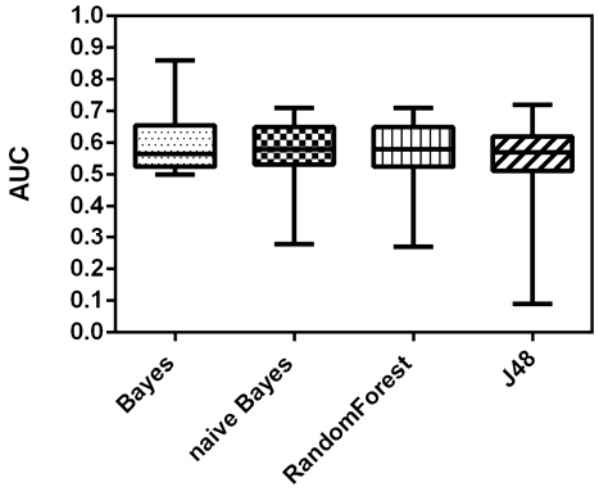
BÖLÜM 7. SONUÇLAR VE ÖNERİLER

7.1. Bilimsel Bulgular

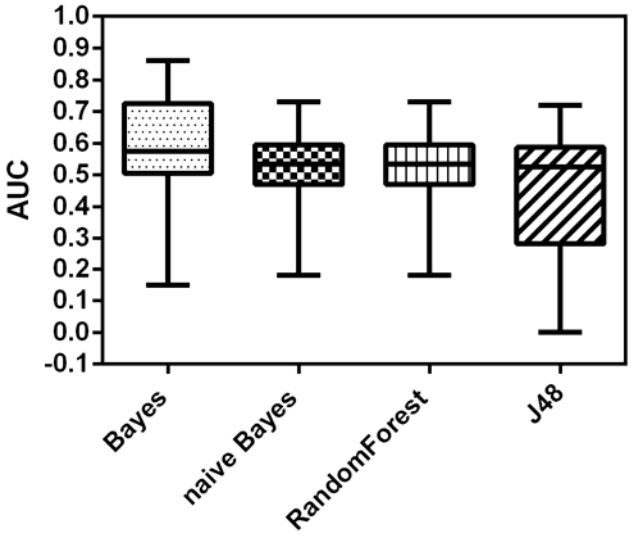
Ön-işleme algoritmasıın performans karşılaştırması için dört farklı sınıflandırıcı Bayes, Naive Bayes, Rastgele Orman ve J48 algoritmaları seçilmiştir. Bu algoritmalarda yapılan sınıflandırma 10*10 çapraz-onaylama ile elde edilmiştir. Performans parametreleri olarak AUC ve Kesinlik seçilmiştir. Tekrar eden veri oranları açık kaynak kodlu projelerde %20 ile %24 arasında değişmektedir. Bu sonuçlar örnek adetlerine bakılmaksızın elde edilmiştir. Tekrar eden veri oranları proje büyüklüklerine bağlı olmaksızın proje tipine bağlı olarak değişmektedir.

Endüstriyel veri setlerinde tekrar eden veri oranları benzerlik göstermezken C++ ile yazılan kc1 ve kc2 projeleri istisnadır. Bu projelerde örnek adedi farkı 1500'dür. Sonuçlar göstermektedir ki kodlama dili ve geliştirici gibi faktörler tekrar eden veri oranlarını etkilemektedir.

Şekil 7.1 tüm sınıflandırıcıların ön-işleme öncesi ve sonrası AUC değerlerinin box-plot grafiklerini göstermektedir. Her box-plot için, ortadaki siyah çizgi AUC değerinin ortalamasını simgeler. Box-plot grafiklerinin en alt ve en üst noktaları sırasıyla en kötü ve en iyi AUC değerini gösterir. Böyle grafiklerde kutunun kısa olması istenir. AUC varyansları karşılaştırıldığında Şekil 7.1-a grafiğinde ön-işleme sonrası daha tutarlı sonuçlar elde edildiği görülmektedir. Sonuç değer aralığı 0.2-0.7 iken, 0.5-0.7 aralığına J48 algoritmasında getirilmiştir. Bu sonuç tüm sınıflandırıcılar arasında J48'de en iyi olarak değerlendirilebilir. AUC değerlerinin detayları Tablo 7.1-7.2'de hata oranlarıyla birlikte görülmektedir. Tablo 7.3 kesinlik performans parametresindeki değişimleri ön-işleme sonrasında değerleriyle birlikte sunmaktadır.



a) Ön-işlem sonrası



b) Ön-işlem öncesi

Şekil 7.1. 20 veri seti üzerindeki sınıflandırıcıların AUC box-plot sonuçları

Ön-işleme ve metriklerin türetilmesi Kesinlik parametresi açısından %6 oranında genelde bir artış sağlamıştır. Diğer taraftan AUC performans parametresinde toplamda +0.14 oranında artış sağlanmıştır. Tahmin başarısının en iyi elde edildiği proje equinox ve sınıflandırıcı Rastgele Orman olarak tespit edilmiştir.

Tablo 7.1. Ön-işleme öncesi AUC değerleri

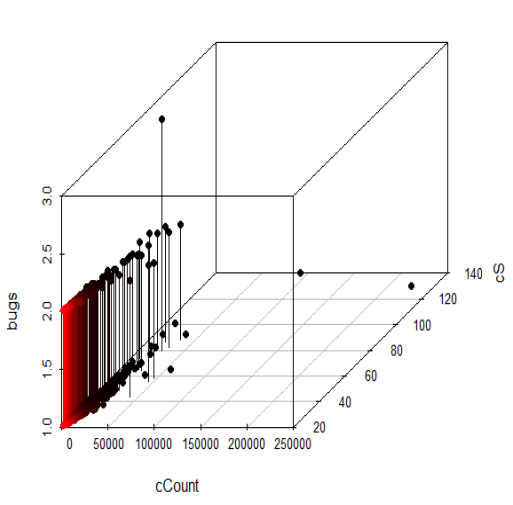
Veri Seti	Bayes	Naive Bayes	Rastgele Orman	J48
eclipsePde	0.57±0.01	0.5±0.01	0.5±0.01	0.51±0.01
equinox	0.58±0.03	0.55±0.03	0.55±0.03	0.55±0.03
jdtCore	0.50±0.02	0.54±0.02	0.54±0.02	0.53±0.02
lucene	0.52±0.02	0.51±0.02	0.51±0.02	0.51±0.02
mylyn	0.53±0.01	0.50±0.01	0.5±0.01	0.52±0.01
ar1	0.5±0.175	0.683±0.176	0.68±0.017	0.5±0.175
ar3	1±0	1±0	1±0	0.5±0.3
ar4	0.66±0.122	0.7±0.11	0.7±0.11	0.68±0.12
ar5	0.8±0.138	0.7±0.15	0.7±0.15	0.78±0.14
ar6	0.5±0.131	0.58±0.133	0.58±0.133	0.58±0.13
cm1	0.542±0.075	0.58±0.07	0.58±0.07	0.54±0.07
jm1	0.44±0.013	0.18±0.011	0.18±0.011	0.16±0.01
kc1	0.582±0.002	0.6±0.02	0.6±0.02	0.59±0.02
kc2	0.727±0.05	0.65±0.05	0.65±0.05	0.71±0.05
kc3	0.15±0.07	0.35±0.09	0.35±0.09	0.28±0.08
pc1	0.722±0.05	0.46±0.06	0.46±0.06	0.29±0.06
pc2	0.75±0.08	0.31±0.13	0.31±0.13	0-0
pc3	0.732±0.03	0.53±0.04	0.53±0.04	0.2±0.03
pc4	0.647±0.04	0.68±0.03	0.68±0.03	0.72±0.03
pc5	0.86±0.01	0.73±0.02	0.73±0.02	0.65±0.02

Tablo 7.2. Ön-işleme sonrası AUC değerleri

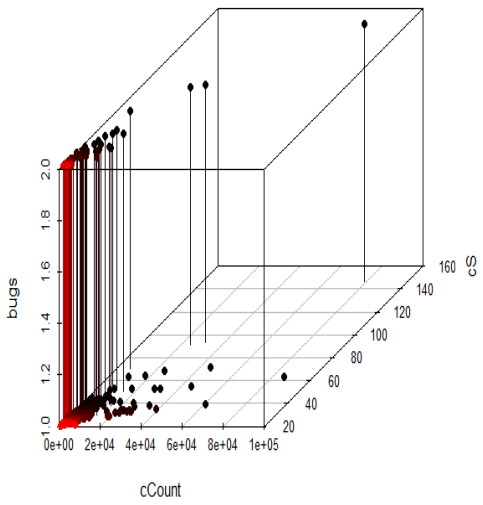
Veri Seti	Bayes	Naive Bayes	Rastgele Orman	J48
eclipsePde	0.55±0.02	0.52±0.02	0.52±0.02	0.52±0.02
equinox	0.56±0.04	0.59±0.04	0.59±0.04	0.55±0.04
jdtCore	0.51±0.02	0.54±0.02	0.54±0.02	0.53±0.02
lucene	0.55±0.03	0.52±0.03	0.52±0.03	0.51±0.03
mylyn	0.52±0.01	0.51±0.01	0.51±0.01	0.5±0.01
ar1	0.5±0.175	0.6±0.176	0.68±0.176	0.5±0.175
ar3	1±0	1±0	1±0	0.5±0.3
ar4	0.66±0.122	0.77±0.111	0.77±0.11	0.68±0.12
ar5	0.8±0.138	0.7±0.158	0.7±0.15	0.78±0.14
ar6	0.5±0.131	0.563±0.133	0.563±0.133	0.58±0.13
cm1	0.514±0.09	0.279±0.09	0.27±0.094	0.09±0.06
jm1	0.794±0.01	0.569±0.01	0.569±0.01	0.58±0.12
kc1	0.571±0.02	0.61±0.02	0.61±0.02	0.59±0.02
kc2	0.715±0.051	0.65±0.05	0.65±0.05	0.71±0.05
kc3	0.64±0.08	0.65±0.08	0.65±0.08	0.63±0.08
pc1	0.54±0.012	0.56±0.01	0.56±0.01	0.58±0.012
pc2	0.66±0.094	0.63±0.09	0.63±0.09	0.5±0.09
pc3	0.59±0.0376	0.67±0.03	0.67±0.03	0.56±0.03
pc4	0.64±0.04	0.68±0.03	0.68±0.03	0.72±0.03
pc5	0.86±0.01	0.71±0.02	0.71±0.02	0.65±0.02

Tablo 7.3. Ön-işleme sonrası Kesinlik değişimleri

Veri Seti	Bayes	Naive Bayes	RastgeleOrman	J48
eclipsePde	-1	0	-7	-1
equinox	+6	+3	-1	-3
jdtCore	+2	+1	0	-1
lucene	+2	+2	+7	+4
mylyn	+3	-1	-7	-3
ar1	-3	+2	+7	-12
ar3	+15	+5	0	0
ar4	+8	+17	+9	+9
ar5	0	0	0	0
ar6	0	0	+3	-3
cm1	-9	+6	+2	-1
jm1	-3	0	+2	-6
kc1	0	0	0	0
kc2	-4	0	+5	+1
kc3	-8	+3	+12	+4
pc1	+22	+3	-8	-1
pc2	+2	-4	+14	+4
pc3	-3	+24	-1	-11
pc4	-2	-5	0	-8
pc5	-1	0	0	0

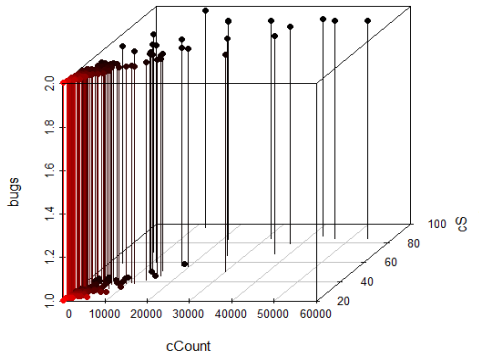


(a) mylyn scatterplot (bugs-cCount-cS).

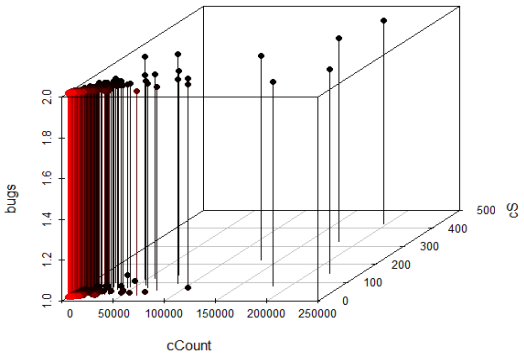


(b) lucene scatterplot (bugs-cCount-cS)

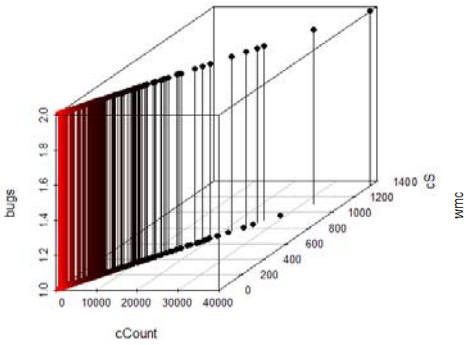
Şekil 7.2. Tüm veri setlerinde dört metriğin scatterplot analizi, alt şekil (a), (b), (c), (d) ve (e) bugs-cCount-cS ilişkisini çizerken, alt şekil (f), (g), (h), (i) ve (j) ise wmc-cCount-cS ilişkisini çizmektedir (Tüm şekiller R istatistiksel analiz paketi kullanılarak oluşturulmuştur [137])



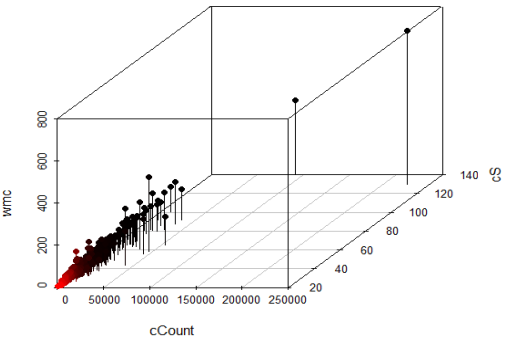
(c) equinox scatterplot (bugs-cCount-cS)



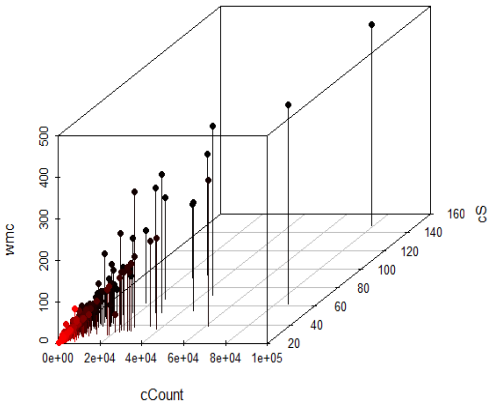
(d) jdtCore scatterplot (bugs-cCount-cS)



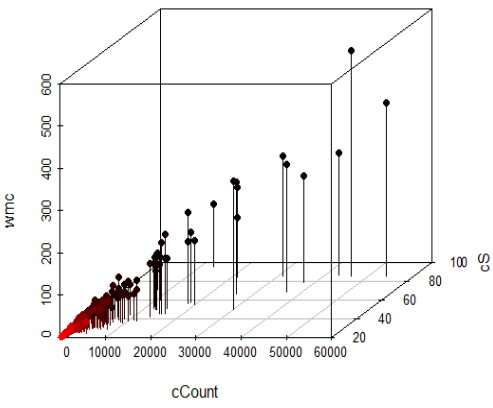
(e) eclipsePde scatterplot (bugs-cCount-cS)



(f) mylyn scatterplot (wmc-cCount-cS)

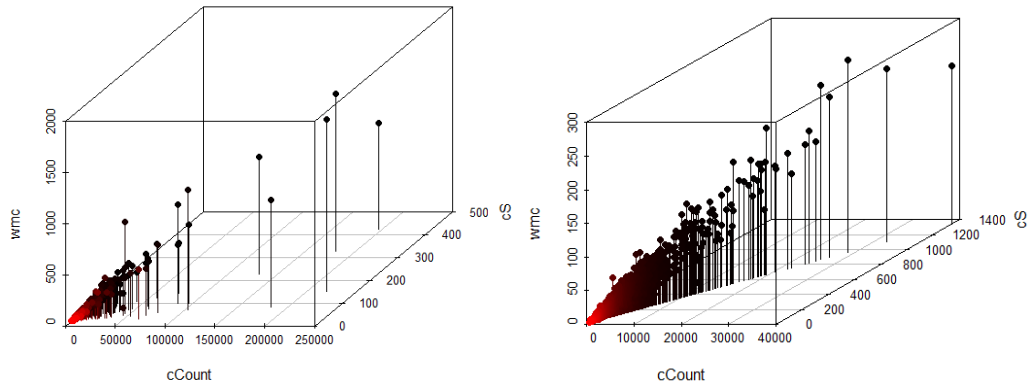


(g) lucene scatterplot (wmc-cCount-cS)



(h) equinox scatterplot (wmc-cCount-cS)

Şekil 7.2. (Devamı)



(i) jdtCore scatterplot (wmc-cCount-cS) (j) eclipsePde scatterplot (wmc-cCount-cS)

Şekil 7.2. (Devamı)

WMC bir kod sınıfının geliştirilmesi için gerekli zaman ve çabayı tahmin eden nesne-yönelimli metriklerden biridir. WMC değeri arttıkça hata yoğunluğu artar ve yazılım kalitesi azalır. WMC metriğini en uygun hale getirmek için tüm sınıfların en az %10 oranında 24'den fazla metot kullanılabilir. Bir diğer yol ise metot adetlerini 20, 50 gibi miktarlarla sınırlamaktır. Şekil 7.2'de verilen scatterplot analizlerine bakarak equinox projesinde hata dağılımlarının en uygun olduğu görülmektedir. Aynı zamanda en iyi AUC artışı equinox projesinde elde edilmiştir (Şekil 7.2 (c)-(h)). Diğer taraftan en köt AUC iyileşmesinin elde edildiği proje mylyn projesidir. Bu projeye ait scatterplot analizleri Şekil 7.2 (a) ve (f)'de görülmektedir. Sonuç olarak fonksiyon dağılımları ne kadar iyi ayarlanırsa hata tahmin performansı o kadar iyi elde edilebilmektedir.

7.2. Geçerlilik için Tehditler

Önerilen yöntemin geçerliliği için bazı tehditleri tartışmamız gerekmektedir. İlk olarak statik kod metrikleri üzerinden metrik türetimi umut verici sonuçlar üretmiştir. Bununla beraber son yıllarda hata tahmininde statik kod metriklerine üstün gelen süreç metrikleri üzerinde de yöntemin denenmesi çalışmanın genişletilebilmesi açısından önemlidir. Genişletilen çalışma statik kod metriklerinde elde edilen sonuçlarla karşılaştırılmalıdır.

Tüm veriler metrik türetimi sonrası ön-işlemeye tabi tutulmuştur. Düşük seviyeli

metrik türetimi tahmin başarısını arttırmıştır ancak yeni metrikler türetilerek hükmün genelliği doğrulanmalıdır. Bu bağlamda türetim işlemi açık kaynak kodlu projelerde yapılırken manuel inceleme ile kontrol edilirse metodun geçerliliği test edilebilir.

10*10 çapraz-onaylama yöntemi kullanılarak sınıflandırma işlemi tamamlanmıştır. Bu yönteme alternatif olarak veri seti ayrımını %66 eğitim, %33 test şeklinde yaparak karşılaştırma yapılabilir. Sonuçları genellemenin bir başka yolu da çapraz-proje veri setleri üzerinde de yöntemi denemektir.

7.3. HSDD-SMOTE-Virtual Kıyaslama

Hata tahmin veri setlerinde örnek-çoğaltma için kullanılan SMOTE algoritması ile bu algoritmaya alternatif Virtual algoritması ve bizim geliştirdiğimiz HSDD algorirmasını g-mean performans parametresi açısından karşılaştırdık. G-mean parametresi genellikle ön-işleme algoritmalarının performans değerlendirmesinde sıklıkla kullanılan bir algoritmadır. Deneysel veri setlerinden pc5, lucene, jm1 ve kc1 veri setleri seçilerek HSDD, SMOTE ve Virtual algoritmaları ile bu veri setlerinden türetilen 100, 200, 300, 400 ve 500 verideki eğitim ile yapılan sınıflandırmada Bayes, Naive Bayes, Rastgele Orman ve J48 algoritmalarının ortalama g-mean değerleri hesaplanarak sonuçlar üretilmiştir.

Ön-işleme algoritmamız ile Virtual ve SMOTE algoritmalarının karşılaştırılabilmesi için K-komşuluk fonksiyonu kullanılması gerekmektedir. Bu fonksiyon Virtual algoritması adımlarında da yer almaktadır.

```
[Serializable]
public class KNearestNeighbors : KNearestNeighbors<double[]>
{
    public KNearestNeighbors(int k, double[][] inputs, int[] outputs);
    public KNearestNeighbors(int k, int classes, double[][] inputs, int[] outputs);
    public KNearestNeighbors(int k, int classes, double[][] inputs, int[] outputs, Func<double[],

    public override int Compute(double[] input, out double[] scores);
}
```

Şekil 7.3. Komşuluk sınıf yapısı

K-komşuluk sınıfı kullanılırken 3 parametre (constructor) almaktadır. Birincisi k adedi. Bu adet komşuluk kararı verilecek en yakın komşulardır. Giriş örnekleri "inputs" ile verilmekte ve 4 adet sınıfa ait yapı kurulmaktadır. Bu sınıf adetleri "outputs" parametresindeki veri grup adedine bakılarak anlaşılabilir. Aşağıdaki girişe bakılarak "inputs" 10 örnek ve 4 sınıf içerdiği görülmektedir.

```
int[] outputs =
                {
                                0, 0,        // İlk iki 0. sınıfından
                                1, 1, 1, 1,  // Sıradaki 4, 1. sınıfından
                                2, 2, 2,     // Sıradaki 3, 2. sınıfından
                                3,           //Son 1, 3. sınıfından
                };
```

4 sınıf

```
KNearestNeighbors knn = new KNearestNeighbors(k: 4, classes: 4,inputs: orneklerGiden, outputs: outputs);
```

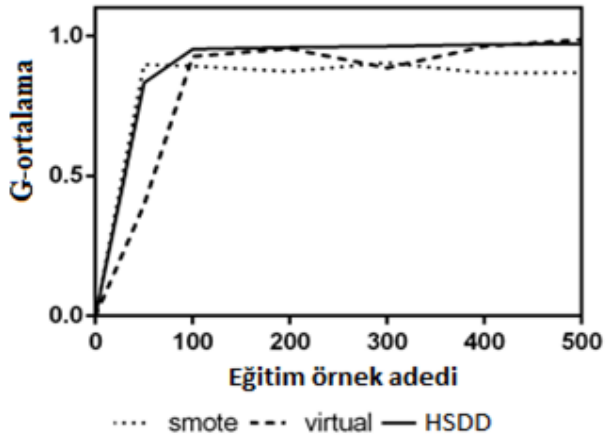
```
-6, 0, 0, 0, -6, -7, -4, 7, -9, -11, -16, -8, 0.5, -2, -76, 0.01, -4.22, 0, 0, 0, 0, 0, -1, 0.17, 0, 0, 0, 0.17, 0, -180, -39,
-9, 0, 0, 0, -16, -17, -4, 7, -9, -11, -16,-38, 0.5, 2, -76, 0.01, -4.22, 0, 0, 0, 0, 0, -1, 0.17, 0, 0, 0, 0.17, 0, -110, -39,
 6, 0, 0, 0, 6, 7, 4, 7, 9, 11, 16, 38, 0.5, 2, 76, 0.01, 4.22, 0, 0, 0, 0, 0, 1, 0.17, 0, 0, 0, 0.17, 0, 180, 39 ,
 6, 0, 0, 0, 6, 7, 4, 7, 9, 11, 16, 38, 0.5, 2, 76, 0.01, 4.22, 0, 0, 0, 0, 0, 1, 0.17, 0, 0, 0, 0.17, 0, 180, 39 ,
 6, 0, 0, 0, 6, 7, 4, 7, 9, 11, 16, 38, 0.5, 2, 76, 0.01, 4.22, 0, 0, 0, 0, 0, 1, 0.17, 0, 0, 0, 0.17, 0, 180, 39,
 6, 0, 0, 0, 6, 7, 4, 7, 9, 11, 16, 38, 0.5, 2, 76, 0.01, 4.22, 0, 0, 0, 0, 0, 1, 0.17, 0, 0, 0, 0.17, 0, 180, 39 ,
 6, 0, 0, 0, 6, 7, 4, 7, 9, 11, 16, 38, 0.5, 2, 76, 0.01, 4.22, 0, 0, 0, 0, 0, 1, 0.17, 0, 0, 0, 0.17, 0, 180, 39,
 6, 0, 0, 0, 6, 7, 4, 7, 9, 11, 16, 38, 0.5, 2, 76, 0.01, 4.22, 0, 0, 0, 0, 0, 1, 0.17, 0, 0, 0, 0.17, 0,     6, 0, 0, 0, 6, 7, 4,
7,
 9, 11, 16, 38, 0.5, 2, 76, 0.01, 4.22, 0, 0, 0, 0, 0, 1, 0.17, 0, 0, 0, 0.17, 0, 180, 39,
  6, 0, 0, 0, 6, 7, 4, 7, 9, 11, 16, 38, 0.5, 2, 76, 0.01, 4.22, 0, 0, 0, 0, 0, 1, 0.17, 0, 0, 0, 0.17, 0, 180, 39,
```

10 satır veri

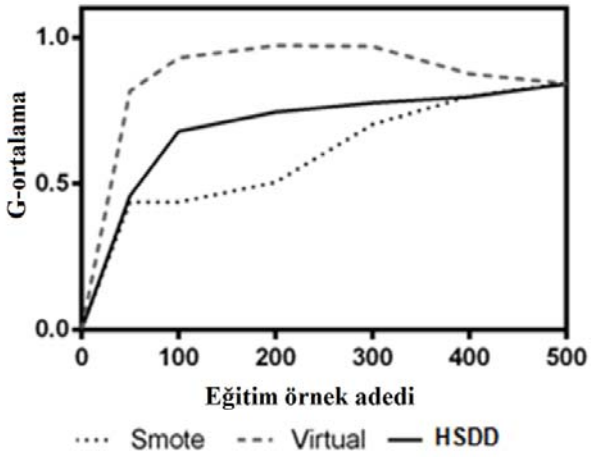
G-mean grafiklerini yorumlamamız gerekirse öncelikle Şekil 7.4'teki pc5 veri seti eğrilerini inceleyelim. Bu veri setinde eğitim veri adedine bakılmaksızın tüm aralıklarda HSDD'nin yakınsaması daha iyidir. Veri seti C++ ile yazılmıştır ve 17186 adet örnek içermektedir. Bununla beraber özellik sayısı 39'dur. Dolayısıyla çok geniş bir ölçekten küçük oranlarda deneme verisinin seçilmiş olması grafik değerlerinin daha doğru elde edilmesine neden olmuş olabilir.

Lucene veri seti Java ile yazılmış bir modülden çıkarılmıştır. Ancak özellik sayısı bu veri setinde 21, örnek sayısı 691'dir. Bu veri setine Şekil 7.5'teki grafiğe bakılarak denebilir ki virtual algoritması bizim önerdiğimiz algoritmaya göre küçük ölçekli veri setlerine daha uygun bir algoritmadır. Burada SMOTE algoritması en kötü eğriyi

üretmiştir.

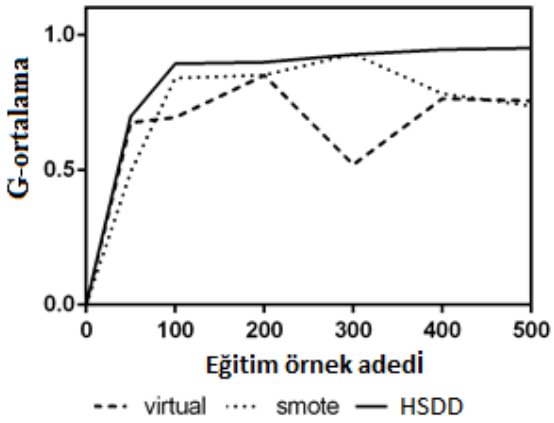


Şekil 7.4. pc5 veri setinde g-mean sonuçları

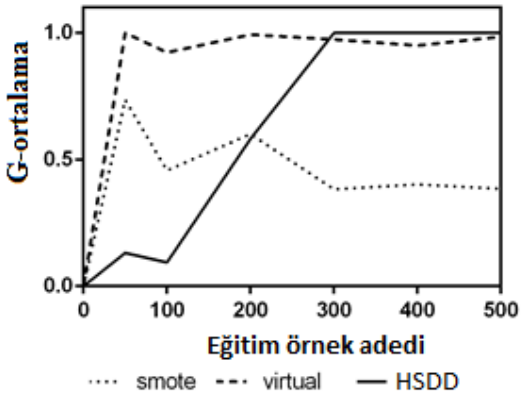


Şekil 7.5. Lucene veri setinde g-mean sonuçları

Kc1 veri setinde elde edilen g-mean eğrilerine baktığımızda (Şekil 7.6) önerdiğimiz algoritmanın en iyi yakınsamayı ürettiğini görüyoruz. Yakınsamanın iyi olması eğitim veri seti adedi değiştiği durumlarda algoritmanın tutarlı g-ortalama değeri ürettiği anlamına gelir. Bu veri seti C++ ile yazılmış bir projeden alınmış olup özellik sayısı 21, örnek sayısı 2107 dir. Önceki grafiklerle karşılaştırdığımızda özellik sayısının azalmasına rağmen örnek sayısının 1000'li değerlerin üzerine çıktığında gene daha yorumlanabilir grafiklerin üretildiği görülmektedir



Şekil 7.6. Kc1 veri setinde g-ortalama sonuçları



Şekil 7.7. Jm1 veri setinde g-ortalama sonuçları

jm1 veri setinde (Şekil 7.7) 300 eğitim adedine gelininceye kadar HSDD en kötüdür. Ancak 300 örnek adedi üzerinde en iyi sonuçları üretmiştir. jm1 veri seti C dilinde yazılmış 21 özellikli 10878 örnek veri içermektedir. kc1 de üretilen iyi sonuçlara rağmen burada düşük eğitim adetlerinde önerdiğimiz yöntemin diğer algoritmalardan daha kötü sonuçlar üretmesini nesne-yönelimli prensiplere bağlayabiliriz. Bizim ön-işleme yöntemimizde yer alan metrik türetimi nesne-yönelimli programlama prensiplerine daha uygundur. Dolayısıyla grafikte elde edilen sonuçlar normaldir.

7.4. Sonuçların Özeti ve Gelecek Çalışmalar

Bu tez, düşük seviyeli metrik türetiminin ve tekrar eden veri analizinin hata tahmini başarısı üzerindeki etkisini araştırmaktadır. Düşük seviyeli metrik kullanımı yazılım sistemlerinin daha iyi anlaşılmasını sağlamaktadır. Bununla beraber, sürekli bir metrik

türetimi belirli bir noktadan sonra tahmin başarısını lineer olarak arttırmamaktadır. Açık kaynak kodlu projelerdeki tekrar eden veri oranları birbirine yakındır. Aksine, endüstriyel veri setlerindeki tekrar eden veri oranları benzer değildir. Bunun nedeni endüstriyel projelerin kodlama ve büyüklük özelliklerinin değişken olması olabilir.

En umut vadeden deneysel veri seti jm1 veri setidir. Bu veri setinde AUC performans parametresi açısından en iyi sonuç elde edilmiştir. Tüm sınıflandırıcılar bu veri setinde tahmin başarılarını arttırmışlardır. En yüksek artış j48 sınıflandırıcısında %0.42 ile elde edilmiştir. AUC ve Kesinlik parametreleri açık kaynak kodlu veri setlerinde sırasıyla %14 ve %6 oranında iyileştirilmiştir. Endüstriyel veri setleri eklendikten sonra bu oranlar %4.05 ve %6.7 olarak değişmiştir. Sonuç olarak HSDD algoritması iki değerlendirme parametresinde açık kaynak kodlu projelerde daha iyi sonuçlar üretmiştir.

WMC metrik değeri azaldıkça tahmin performansı artmıştır. Nitekim sınıflardaki metot dağılımı bu açıdan kritiktir. Düşük seviyeli metriklerin tahmin başarısına etkisi süreç metrikleri açısından araştırılmamıştır. Gelecek çalışmalarda son yıllarda umut verici sonuçlar üreten süreç metriklerinde, düşük seviyeli metrik türetim yöntemleri araştırılmalıdır.

Süreç metriklerin son yıllarda umut verici sonuçlar ürettiği görülmektedir. Dolayısıyla önerdiğimiz HSDD gelecek çalışmalarda süreç metriklerinde denenmelidir. Bunun dışında bulanık mantık tabanlı yöntemler hata veri setlerindeki gürültü verilerin tespitinde kullanılabilir. HSDD'nin karar mekanizmasına istatistiksel yöntemlerin yanında bulanık mantık kullanılarak yeni bir karar mekanizması eklenebilir.

HSDD 20 veri seti üzerinde uygulanmış ve %17-%24 arasında değişen oranlarda tekrar eden verileri tespit etmiştir. Tahmin başarısı bu veriler silindikten sonra tekrar ölçülmüştür. Metodun veri madenciliğinde alternatif bir algoritma olarak kullanılabilir olduğunun onaylanması için veri seti tipleri genişletilmelidir.

KAYNAKLAR

[1] Lee, C. K. H., Choy, K. L. K., Ho, G. T. S., Chin, K. S., Law, K. M. Y., Tse, Y. K., A hybrid OLAP-association rule mining based quality management system for extracting defect patterns in the garment industry. Expert Systems with Applications, 40 (7), 2435–2446, 2013.

[2] D'Ambros, M., Lanza, M., Robbes, R., Romain, D'Ambros, M., Lanza, M., Robbes, R., D'Ambros, M., Lanza, M., Robbes, R., Evaluating defect prediction approaches: a benchmark and an extensive comparison. Empirical Software Engineering, 17 (4)–(5), 531–577, 2012.

[3] Li, M., Zhang, H., Wu, R., Zhou, Z.-H., Sample-based software defect prediction with active and semi-supervised learning. Automated Software Engineering, 19 (2), 201–230, 2012.

[4] Bettenburg, N., Nagappan, M., Hassan, A. E. A., Think locally, act globally: Improving defect and effort prediction models. IEEE International Working Conference on Mining Software Repositories , 60–69, 2012.

[5] Sun, Z., Song, Q., Zhu, X., Using coding-based ensemble learning to improve software defect prediction. IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews, 42 (6), 1806–1817, 2012.

[6] McCabe, T. J., A Complexity Measure. IEEE Transactions on Software Engineering, SE-2 (4), 308–320, 1976.

[7] Chidamber, S. R., Kemerer, C. F., A metrics suite for object oriented design. IEEE Transactions on Software Engineering, 20 (6), 476–493, 1994.

[8] Lorenz, M., Kidd, J., Object-Oriented Software Metrics. Journal of Systems and Software, 44 (2), 147–154, 1994.

[9] Liu, H., Building effective defect-prediction models in practice. Software, IEEE, 22 (6), 23–29, 2005.

[10] Menzies, T., Milton, Z., Turhan, B., Cukic, B., Jiang, Y., Bener, A., Defect prediction from static code features: current results, limitations, new approaches. Automated Software Engineering, 17 (4), 375–407, 2010.

[11] Lessmann, S., Baesens, B., Mues, C., Pietsch, S., Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel

Findings. IEEE Transactions on Software Engineering, 34 (4), 485–496, 2008.

[12] Shepperd, M., Bowes, D., Hall, T., Researcher Bias : The Use of Machine Learning in Software Defect Prediction. Software Engineering, IEEE Transactions on, 40 (6), 603–616, 2014.

[13] Khoshgoftaar, T., An empirical study of feature ranking techniques for software quality prediction. International Journal of Software Engineering and Knowledge Engineering, 22 (2), 161–183, 2012.

[14] Kim, S., Zhang, H., Wu, R., Gong, L., Dealing with noise in defect prediction. 2011 33rd International Conference on Software Engineering (ICSE), 481–490, 2011.

[15] Wang, S., Yao, X., Using class imbalance learning for software defect prediction. IEEE Transactions on Reliability, 62 (2), 434–443, 2013.

[16] Attenberg, J., Ertekin, S., Class Imbalance and Active Learning. Imbalanced Learning: Foundations, Algorithms, and Applications, 101–149, 2013.

[17] Menzies, T., Greenwald, J., Frank, A., Data mining static code attributes to learn defect predictors. Software Engineering, IEEE Transactions on, 31 (1), 2–13, 2007.

[18] Jiang, Y., Cuki, B., Menzies, T., Bartlow, N., Cukic, B., Menzies, T., Bartlow, N., Comparing design and code metrics for software quality prediction. Proceedings of the 4th international workshop on Predictor models in software engineering PROMISE 08, 12, 11–18, 2008.

[19] Rahman, F., Devanbu, P., Ownership, experience and defects. Proceeding of the 33rd international conference on Software engineering - ICSE '11 , 491, 2011.

[20] Madeyski, L., Jureczko, M., Which process metrics can significantly improve defect prediction models? An empirical study. Software Quality Journal, 23 (3), 393–422, 2014.

[21] Laradji, I. H. I., Alshayeb, M., Ghouti, L., Software defect prediction using ensemble learning on selected features. Information and Software Technology, 58, 388–402, 2015.

[22] Menzies, T., Kocaguneli, E., Turhan, B., Minku, L., Peters, F., Sharing Data and Models in Software Engineering: Sharing Data and Models. , 2014.

[23] Gray, D., Bowes, D., Davey, N., Sun, Y., Christianson, B., Reflections on the NASA MDP data sets. IET Software, 6 (6), 549, 2012.

[24] Siers, M. M. J., Islam, M. M. Z., Software defect prediction using a cost

sensitive decision forest and voting and a potential solution to the class imbalance problem. Information Systems, 51, 62–71, 2015.

[25] D'Ambros, M., Lanza, M., Robbes, R., An extensive comparison of bug prediction approaches. Mining Software Repositories (MSR), 31–41, 2010.

[26] Internet: Menzies, T., Rees-Jones, M., Krishna, R., Pape, C., Pryor, D., The Promise Repository of Empirical Software Engineering Data. http://openscience.us/repo .

[27] Song, Q., Jia, Z., Shepperd, M., Ying, S., Liu, J., A general software defect-proneness prediction framework. IEEE Transactions on Software Engineering, 37 (3), 356–370, 2011.

[28] Koru, a. G., El Emam, K., Zhang, D., Liu, H., Mathew, D., Theory of relative defect proneness: Replicated studies on the functional form of the size-defect relationship. Empirical Software Engineering, 13 (October 2015), 473–498, 2008.

[29] Cartwright, M., Shepperd, M., An empirical investigation of an object-oriented software system. IEEE Transactions on Software Engineering, 26 (8), 786–796, 2000.

[30] Zhou, Y., Leung, H., Society, I. C., Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults. Software Engineering, IEEE Transactions on, 32 (10), 771–789, 2006.

[31] Nair, T. R. G., Selvarani, R., Defect proneness estimation and feedback approach for software design quality improvement. Information and Software Technology, 54 (3), 274–285, 2012.

[32] Yu, L., Mishra, A., Experience in Predicting Fault-Prone Software Modules Using Complexity Metrics. Quality Technology and Quantitative Management, 9 (4), 421–433, 2012.

[33] Catal, C., Sevim, U., Diri, B., Practical development of an Eclipse-based software fault prediction tool using Naive Bayes algorithm. Expert Systems with Applications, 38 (3), 2347–2353, 2011.

[34] Turhan, B., Bener, A., A Multivariate Analysis of Static Code Attributes for Defect Prediction. Seventh International Conference on Quality Software (QSIC 2007), (Qsic), 231–237, 2007.

[35] Turhan, B., Menzies, T., Bener, A. B. A., Di Stefano, J., Stefano, J. Di, On the relative value of cross-company and within-company data for defect prediction. Empirical Software Engineering, 14 (5), 540–578, 2009.

[36] Guo, L., Ma, Y., Cukic, B., Singh, H., Guo, L. and Yan Ma and Cukic, Bojan

and Singh, H., Robust Prediction of Fault-Proneness by Random Forests. 15th International Symposium on Software Reliability Engineering, 417–428, 2004.

[37] Challagulla, V., Bastani, F., Empirical assessment of machine learning based software defect prediction techniques. International Journal on Artificial Intelligence Tools, 17 (2), 389–400, 2008.

[38] Zimmermann, T., Nagappan, N., Gall, H., Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM, 91–100, 2009.

[39] He, Z., Shu, F., Yang, Y., Li, M., Wang, Q., An Investigation on the Feasibility of Cross-Project Defect Prediction. Automated Software Engineering, 167-199 , 2012.

[40] Rahman, F., Posnett, D., Devanbu, P., Recalling the imprecision of cross-project defect prediction. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering , 61, 2012.

[41] Ma, Y., Luo, G., Zeng, X., Chen, A., Transfer learning for cross-company software defect prediction. Information and Software Technology, 54 (3), 248–256, 2012.

[42] Kamei, Y., Fukushima, T., McIntosh, S., Yamashita, K., Ubayashi, N., Hassan, A. E., Studying Just-in-Time Defect Prediction Using Cross-Project Models. Empirical Software Engineering, Empirical Software Engineering , 2015.

[43] Mizuno, O., Hirata, Y., A Cross-Project Evaluation of Text-Based Fault-Prone Module Prediction. 2014 6th International Workshop on Empirical Software Engineering in Practice , 43–48, 2014.

[44] Chowdhury, I., Zulkernine, M., Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. Journal of Systems Architecture, 57 (3), 294–313, 2011.

[45] Malhotra, R., A systematic review of machine learning techniques for software fault prediction. Applied Soft Computing, 27, 504–518, 2015.

[46] Foucault, M., Teyton, C., Lo, D., Blanc, X., Falleri, J.-R., On the usefulness of ownership metrics in open-source software projects. Information and Software Technology, 64, 102–112, 2015.

[47] Pears, R., Finlay, J., Connor, A., Synthetic Minority Over-sampling TEchnique (SMOTE) for predicting software build outcomes. ArXiv preprint arXiv:1407.2330, 2014.

[48] Kim, D., Tao, Y., Kim, S., Zeller, A., Where should we fix this bug? A two-

phase recommendation model. IEEE Transactions on Software Engineering, 39 (11), 1597–1610, 2013.

[49] Shatnawi, R., Empirical study of fault prediction for open-source systems using the Chidamber and Kemerer metrics. IET Software, 8 (3), 113–119, 2014.

[50] Shepperd, M., Song, Q., Data quality: Some comments on the nasa software defect datasets. Software Engineering, IEEE Transactions on, 39 (9), 1208–1215, 2013.

[51] Liebchen, G., Shepperd, M., Data sets and data quality in software engineering. In Proceedings of the 4th international workshop on Predictor models in software engineering , 39–44, 2008.

[52] Internet: Rajesh Vasa Markus Lumpe, Jones, A., Helix - Software Evolution Data Set. http://www.ict.swin.edu.au/research/projects/helix .

[53] Keivanloo, I., Forbes, C., A Linked Data platform for mining software repositories. Mining Software Repositories (MSR), 32–35, 2012.

[54] Linstead, E., Bajracharya, S., Ngo, T., Sourcerer: mining and searching internet-scale software repositories. Data Mining and Knowledge Discovery, 18 (2), 300–336, 2009.

[55] Herzig, K., Just, S., Zeller, A., It's not a bug, it's a feature: How misclassification impacts bug prediction. Proceedings - International Conference on Software Engineering, 392–401, 2013.

[56] Fenton, N., Neil, M., Marsh, W., Hearty, P., Marquez, D., Krause, P., Mishra, R., Predicting software defects in varying development lifecycles using Bayesian nets. Information and Software Technology, 49 (1), 32–43, 2007.

[57] Menzies, T., Koru, G., Predictive models in software engineering. Empirical Software Engineering, 18 (3), 433–434, 2013.

[58] Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S., A Systematic Review of Fault Prediction Performance in Software Engineering. IEEE Transactions on Software Engineering, 38 (6), 1276–1304, 2011.

[59] Davis, J., Goadrich, M., The relationship between Precision-Recall and ROC curves. Proceedings of the 23rd international conference on Machine learning, 233–240, 2006.

[60] Ren, J., Qin, K., Ma, Y., Luo, G., On Software Defect Prediction Using Machine Learning. Journal of Applied Mathematics, 2014, 2014.

[61] Singh, R., International Standard ISO/IEC 12207 software life cycle processes. Software Process Improvement and Practice, 2 (1), 35–50, 1996.

[62] Maier, M., Emery, D., Hilliard, R., Software architecture: introducing IEEE Standard 1471. Computer, 2001.

[63] Graham, D., Veenendaal, E. Van, Evans, I., Foundations of software testing: ISTQB certification. Cengage Learning EMEA, 2008.

[64] Humphrey, W., A Discipline for Software Engineering. Addison-Wesley Longman Publishing, 1995.

[65] Posner, E. A., Spier, K. E., Vermeule, A., Divide and Conquer. Journal of Legal Analysis, 2 (2), 417–471, 2010.

[66] Counsell, S., Mendes, E., Swift, S., Comprehension of object-oriented software cohesion: the empirical quagmire. Proceedings 10th International Workshop on Program Comprehension, 33–42, 2002.

[67] Valerdi, R., Wheaton, M., ANSI/EIA 632 as a standardized WBS for COSYSMO. AIAA 1st Infotech@ Aerospace Conference, 2005.

[68] Boegh, J., A new standard for quality requirements. IEEE Software, 25 (2), 57–63, 2008.

[69] Boehm, B., A spiral model of software development and enhancement. Computer, 1988.

[70] Edmonds, E., A process for the development of software for non-technical users as an adaptive system. General Systems, 1974.

[71] Chemuturi, M., Requirements engineering and management for software development projects., 2012.

[72] Schwaber, K., Agile project management with Scrum. Microsoft Press, 2004.

[73] Astels, D., Test Driven Development: A Practical Guide. Prentice Hall Professional Technical Reference, 2003.

[74] Öztürk, M. M., Zengin, A., Improved GUI Testing using Task Parallel Library. ACM SIGSOFT Software Engineering Notes, 41 (1), 1–8, 2016.

[75] Holmes, A., Kellogg, M., Automating Functional Tests Using Selenium. AGILE 2006 (AGILE'06), 270–275, 2006.

[76] Elish, K., Elish, M., Predicting defect-prone software modules using support vector machines. Journal of Systems and Software, 2008.

[77] IEEE Standard Glossary of Software Engineering Terminology. Institute Of Electrical And Electronics Engineers, 1990.

[78] Black, R., Advanced Software Testing-Vol. 2: Guide to the Istqb Advanced Certification as an Advanced Test Manager. Rocky Nook, 2014.

[79] Compton, B. T., Withrow, C., Prediction and control of ADA software defects. Journal of Systems and Software, 12 (3), 199–207, 1990.

[80] Fenton, N. E., Neil, M., Software metrics: successes, failures and new directions. Journal of Systems and Software, 47 (2)–(3), 149–157, 1999.

[81] Curtis, B., Measurement and experimentation in software engineering. Proceedings of the IEEE, 1980.

[82] Nguyen, V., Deeds-Rubin, S., Tan, T., Boehm, B., A SLOC counting standard. COCOMO II Forum, 2007.

[83] Halstead, M. H., Elements of Software Science (Operating and Programming Systems Series). Elsevier Science Inc., 1-400 , 1977.

[84] Jensen, H. A., Vairavan, K., An Experimental Study of Software Metrics for Real-Time Software. IEEE Transactions on Software Engineering, SE-11 (2), 231–234, 1985.

[85] Glasberg, D., El-Emam, K., Memo, W., Madhavji, N., Validating object-oriented design metrics on a commercial java application. TR ERB- 1080, NRC, 2000.

[86] Misra, S. C., Bhavsar, V. C., Relationships Between Selected Software Measures and Latent Bug-Density: Guidelines for Improving Quality. , 724–732, 2003.

[87] Etzkorn, L., Davis, C., Li, W., A statistical comparison of various definitions of the LCOM metric. The University of Alabama in Huntsville, Alabama, 1997.

[88] Śliwerski, J., Zimmermann, T., Zeller, A., When do changes induce fixes? ACM SIGSOFT Software Engineering Notes, 30 (4), 1, 2005.

[89] Williams, C., Spacco, J., Szz revisited: verifying when changes induce fixes. Proceedings of the 2008 workshop on Defects, 32–36, 2008.

[90] Jung, Y., Oh, H., Yi, K., Identifying static analysis techniques for finding non-fix hunks in fix revisions. Proceedings of the ACM first international workshop, 13–18, 2009.

[91] Meneely, A., Srinivasan, H., Musa, A., Tejeda, A. R., Mokary, M., Spates, B., When a Patch Goes Bad: Exploring the Properties of Vulnerability-Contributing Commits. 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, 65–74, 2013.

[92] Serrano, N., Ciordia, I., Bugzilla, ITracker, and other bug trackers. Software, IEEE, 22 (2), 11–13, 2005.

[93] D'Ambros, M., Lanza, M., Robbes, R., An Extensive Comparison of Bug Prediction Approaches. Proceedings of MSR 2010 (7th IEEE Working Conference on Mining Software Repositories), 31–41, 2010.

[94] Kultur, Y., Turhan, B., Bener, A., ENNA: software effort estimation using ensemble of neural networks with associative memory. SIGSOFT '08/FSE-16 Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, 330–338, 2008.

[95] Carbonell, J., Michalski, R., Mitchell, T., An overview of machine learning. Machine learning, 1983.

[96] Alpaydin, E., Introduction to Machine Learning. MIT Press, 2014.

[97] Kyan, M., Guan, L., Jarrah, K., Muneesawang, P., Unsupervised Learning via Self-Organization. Wiley-IEEE Press, 2014.

[98] MacQueen, J., Some methods for classification and analysis of multivariate observations. Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, 281–297, 1967.

[99] Arthur, D., Vassilvitskii, S., K-means++: The advantages of careful seeding. Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, 1027–1035, 2007.

[100] Arthur, D., Vassilvitskii, S., How Slow is the k-means Method? Proceedings of the twenty-second annual symposium on Computational geometry, 144–153, 2006.

[101] Torres, D., Ruiz, J., Rodriguez, Y., An instance based learning model for classification in data streams with concept change. Artificial Intelligence (MICAI), 2012 11th Mexican International Conference on. IEEE, 58–62, 2012.

[102] Freitag, D., Machine learning for information extraction in informal domains. Machine learning, 2000.

[103] Roussopoulos, N., Kelley, S., Vincent, F., Nearest neighbor queries. ACM sigmod record, 1995.

[104] Dudani, S., The distance-weighted k-nearest-neighbor rule. Systems, Man and Cybernetics, IEEE Transactions on, 4, 325–327, 1976.

[105] Hwang, J., Choi, J., Query-based learning applied to partially trained multilayer perceptrons. Neural Networks, IEEE Transactions on, 2 (1), 131–136, 1991.

[106] Chang, R., Hsiao, P., Unsupervised query-based learning of neural networks using selective-attention and self-regulation. Neural Networks, IEEE Transactions on, 1997.

[107] Balcan, M., Long, P., Active and passive learning of linear separators under log-concave distributions. ArXiv preprint arXiv:1211.1082, 2012.

[108] Bennett, K., Mangasarian, O., Robust linear programming discrimination of two linearly inseparable sets. Optimization methods and software, 1 (1), 23–34, 1992.

[109] Aronov, B., Garijo, D., Núnez-Rodrıguez, Y., Measuring the error of linear separators on linearly inseparable data. , 2010.

[110] Mingers, J., An empirical comparison of selection measures for decision-tree induction. Machine learning, 3 (4), 319–342, 1989.

[111] Smith, D., Top-down synthesis of divide-and-conquer algorithms. Artificial Intelligence, 27 (1), 43–96, 1985.

[112] Freund, Y., Mason, L., The alternating decision tree learning algorithm. Icml, 99, 124–133, 1999.

[113] Quinlan, R., C4.5: Programs for Machine Learning. San Francisco, USA, Morgan Kaufmann Publishers Inc, 1993.

[114] Quinlan, J., Improved use of continuous attributes in C4. 5. Journal of artificial intelligence research, 77–90, 1996.

[115] Crawford, S., Extensions to the CART algorithm. International Journal of Man-Machine Studies, 31 (2), 197–217, 1989.

[116] Edwards, A., Commentary on the arguments of Thomas Bayes. Scandinavian Journal of Statistics, 116–118, 1978.

[117] Breiman, L., Random forests. Machine learning, 45 (1), 5–32, 2001.

[118] Menzies, T., Data mining: a tutorial. Recommendation Systems in Software Engineering, 354, 2014.

[119] Breiman, L., Bagging predictors. Machine learning, 24 (2), 123–140, 1996.

[120] Braga, P., Oliveira, A., Bagging predictors for estimation of software project effort. IEEE Neural Networks, Neural Networks, 2007.

[121] Stigler, S., Thomas Bayes's bayesian inference. Journal of the Royal Statistical Society. Series A (, 250–258, 1982.

[122] Androutsopoulos, I., Koutsias, J., An evaluation of naive bayesian anti-spam filtering. Proceedings of the workshop on Machine Learning in the New Information Age, 9–17, 2000.

[123] Zhang, H., Su, J., Naive Bayesian classifiers for ranking. Machine Learning: ECML 2004 , 501–512, 2004.

[124] Boser, B., Guyon, I., Vapnik, V., A training algorithm for optimal margin classifiers. Proceedings of the fifth annual workshop on Computational learning theory, 144–152, 1992.

[125] Scholkopf, B., Smola, A., Learning with Kernels: Support Vector Machines, Regularization, Optimization, and beyond. MIT Press, 2001.

[126] Min, J., Lee, Y., Bankruptcy prediction using support vector machine with optimal choice of kernel function parameters. Expert systems with applications, 28 (4), 603–614, 2005.

[127] Japkowicz, N., Stephen, S., The class imbalance problem: A systematic study. Intelligent data analysis, 6 (5), 429–449, 2002.

[128] Chawla, N., Bowyer, K., SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research, 321–357, 2002.

[129] Ertekin, Ş., Adaptive Oversampling for Imbalanced Data Classification. Information Sciences and Systems 2013, 261–269, 2013.

[130] Shao, J., Linear model selection by cross-validation. Journal of the American statistical Association, 88 (422), 486–494, 1993.

[131] Gribskov, M., Robinson, N., Use of receiver operating characteristic (ROC) analysis to evaluate sequence matching. Computers & chemistry, 20 (1), 25–33, 1996.

[132] Bradley, A., The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern recognition, 30 (7), 1145–1159, 1997.

[133] Flach, P., The geometry of ROC space: understanding machine learning metrics through ROC isometrics. ICML, 194–201, 2003.

[134] Hoaglin, D., Welsch, R., The hat matrix in regression and ANOVA. The American Statistician, 32 (1), 17–22, 1978.

[135] Miller, I., Freund, J., Johnson, R., Probability and statistics for engineers. Prentice-Hall, 1965.

[136] Hauke, J., Kossowski, T., Comparison of values of Pearson's and Spearman's

correlation coefficients on the same sets of data. Quaestiones Geographicae, 30 (2), 87–93, 2011.

[137] Ligges, U., Mächler, M., Scatterplot3d-an r package for visualizing multivariate data. , 2002.

ÖZGEÇMİŞ

Muhammed Maruf ÖZTÜRK 1986 yılında ISPARTA'da doğdu. İlkokulu ve orta okulu Isparta'da tamamladı. Lise öğrenimini Gülkent Lisesinde tamamladı. 2008 yılında Pamukkale Üniversitesi Bilgisayar Mühendisliği bölümünden mezun oldu. 2009-2010 yılları arasında askerlik hizmetini tamamladı. 2010-2011 yılları arasında Keytorc Teknoloji firmasında yazılım test mühendisi olarak görev yaptı. 2011-2012 yıllarında Ter Yazılım firmasında web yazılımcısı olarak görev yaptı. 2012 yılında Sakarya Üniversitesi Bilgisayar ve Bilişim Fakültesi Bilgisayar Mühendisliği Bölümünde yükesek lisans eğitimini tamamladı. Halen, Sakarya Üniversitesi Bilgisayar ve Bilişim Fakültesi Bilgisayar Mühendisliği Bölümünde Araştırma Görevlisi olarak çalışmaktadır.