ANKARA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

NESNE TABANLI METRİKLER KULLANILARAK YAZILIM PROJELERİ MALİYETLERİNİN TAHMİN EDİLMESİ

Adem DİLBAZ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

ANKARA
2020

TEZ ONAYI

Adem DİLBAZ tarafından hazırlanan **"Nesne Tabanlı Metrikler Kullanılarak Yazılım Projeleri Maliyetlerinin Tahmin Edilmesi"** adlı tez çalışması 14/02/2020 tarihinde aşağıdaki jüri tarafından oy birliği ile Ankara Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Danışman : Dr.Öğretim Üyesi Bülent TUĞRUL
Ankara Üniversitesi Bilgisayar Mühendisliği Anabilim Dalı

Jüri Üyeleri :

Başkan: Prof. Dr. Süleyman TOSUN
Hacettepe Üniversitesi Bilgisayar Mühendisliği Anabilim Dalı

Üye: Doç. Dr. Mehmet Serdar GÜZEL
Ankara Üniversitesi Bilgisayar Mühendisliği Anabilim Dalı

Üye: Dr.Öğretim Üyesi Bülent TUĞRUL
Ankara Üniversitesi Bilgisayar Mühendisliği Anabilim Dalı

Yukarıdaki sonucu onaylarım.

Prof. Dr. Özlem YILDIRIM
Enstitü Müdürü

ETİK

Ankara Üniversitesi Fen Bilimleri Enstitüsü tez yazım kurallarına uygun olarak hazırladığım bu tez içindeki bütün bilgilerin doğru ve tam olduğunu, bilgilerin üretilmesi aşamasında bilimsel etiğe uygun davrandığımı, yararlandığım bütün kaynakları atıf yaparak belirttiğimi beyan ederim.

14/02/2020

Adem DİLBAZ

ÖZET

Yüksek Lisans Tezi

NESNE TABANLI METRİKLER KULLANILARAK YAZILIM PROJELERİ MALİYETLERİNİN TAHMİN EDİLMESİ

Adem DİLBAZ

Ankara Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Dr. Öğr. Üyesi Bülent TUĞRUL

Yazılım metrikleri, yazılım projelerinin ölçülebilen ya da yapılan bu ölçümlere göre hesaplanan değerlerine denir. Yazılım metrikleri, yazılımların test, geliştirme, bakım, hata giderme ve proje yönetimi gibi konularda yazılımın birçok yönden değerlendirilmesini sağlayan ölçüm yöntemleridir. Bir yazılım geliştirici maliyet, hata, güvenilirlik, test, güvenlik tahminleri gibi birçok zorlu ihtiyaçlar ile baş etmek durumundadır. Yazılım geliştirmenin büyük oranda payını insan gücü oluşturduğundan yapılan işin her aşamada takibi de önem arz etmektedir. Bu yüksek lisans tez çalışmasında sık kullanılan yazılım metrikleri, bu metriklerin sonuçlarının değerlendirilip analiz edilmesi ve yazılım ölçüm metrikleri ile ilgili çalışmalar anlatılmış, literatür taraması sonrasında 103 farklı Java kütüphanesinin majör versiyonlarına ait kaynak kodlar Github ve Maven Repository kaynaklarından elde edilmiş, bu kodlar Source Monitor adlı yazılım metrik ölçüm aracına girdi olarak verilerek, 13 farklı yazılım metriği bu kütüphaneler için hesaplanmıştır. Metrik sonuçları normalize edilerek, Rapidminer uygulaması kullanılarak 3 farklı makine öğrenme algoritmasına kaynak olarak verilmiş, algoritmaların parametrelerindeki değişime göre hata payı (RMSE) değişimi takip edilmiştir. Bu veriler ışığında yazılım versiyon değişikliğindeki maliyet ölçümlemede minimum hata oranı hangi algoritmada ve hangi parametrelerle gerçekleşebileceği gözlemlenmiştir.

Şubat 2020, 55 sayfa

Anahtar Kelimeler: Yazılım metrikleri, Yazılım Kalitesi, Yazılımda Maliyet Ölçümü, Yazılım Ölçüm Araçları, Yazılım Ölçümlerinde Değerlendirme

ABSTRACT

Master Thesis

PREDICTION OF SOFTWARE PROJECT COSTS USING OBJECT-ORIENTED METRICS

Adem DİLBAZ

Ankara University
Graduate School of Natural and Applied Sciences
Department of Computer Engineering

Supervisor: Dr. Öğr. Üyesi Bülent TUĞRUL

Software metrics are the values of software projects that can be measured or calculated measurements. They are measurement methods enabling software to be evaluated in many ways such as software testing, development, maintenance, troubleshooting, and project management. A software developer has to cope with many demanding needs such as cost, error, reliability, testing, and security estimates. As the majority of software development is manpower, monitoring of the work at every stage is also important. In this master thesis, software metrics that are used frequently, evaluation and analysis of the results of these software metrics and studies related with software measurement metrics are explained. 13 different software metrics from Github and Maven Repository were calculated for these libraries by giving the software as input to the metric measurement tool named Source Monitor. Metric results were normalized and given as a source for 3 different machine learning algorithms by using Rapidminer application. According to this study, it is observed which algorithm and parameters can be realized with minimum error rate (RMSE) in cost measurement in software version change.

February 2020, 55 pages

Keywords: Software metrics, Software Quality, Software Cost Measurement, Software Measurement Tools, Evaluation in Software Measurement

TEŞEKKÜR

Bu araştırma için beni yönlendiren, çalışmalarım süresince ilgisini ve yardımlarını benden esirgemeyen değerli hocam Dr. Öğr. Üyesi Bülent Tuğrul'a,

Sevgisiyle, sabrıyla bana her zaman güç veren, beni bugünlere getiren, en iyi şekilde yetiştiren canım anneme,

Bana olan sonsuz inancını ve güvenini her zaman hissettiğim, sevgisiyle her zaman yanımda olan, bana güç veren canım eşim Gülsüm'e,

İçtenlikle teşekkür ederim.

Adem DİLBAZ
Ankara, Şubat 2020

İÇİNDEKİLER

KISALTMALAR DİZİNİ

DVM	Destek Vektör Makineleri
YSA	Yapay Sinir Ağları
SVM	Support Vector Machines
RMSE	Root Mean Square Error
ANN	Artificial Neural Network
SQM	Software Quality Model
LCOM	Lack of Cohesion in Methods
RFC	Response for Class
DIT	Depth of Inheritance
WMC	Weighted Methods for Class
NOM	Number of Methods
SIX	Specialization Index
COCOMO	Constructive Cost Model

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

1. GİRİŞ

Yazılım endüstrisindeki gelişmeler, yazılım geliştirme sürecini doğrudan etkileyerek bu süreci proje yönetimi çerçevesinde geliştirilen faaliyetler haline getirmiştir. Yazılım projelerinin gitgide büyümesi, karmaşıklaşması ve boyutlarının sürekli artması yazılım kalitesini de etkilemekte, bakım maliyetlerinin zaman ve çaba olarak artması problemlerini de beraberinde getirmektedir (Subramanian and Corbin 2001).

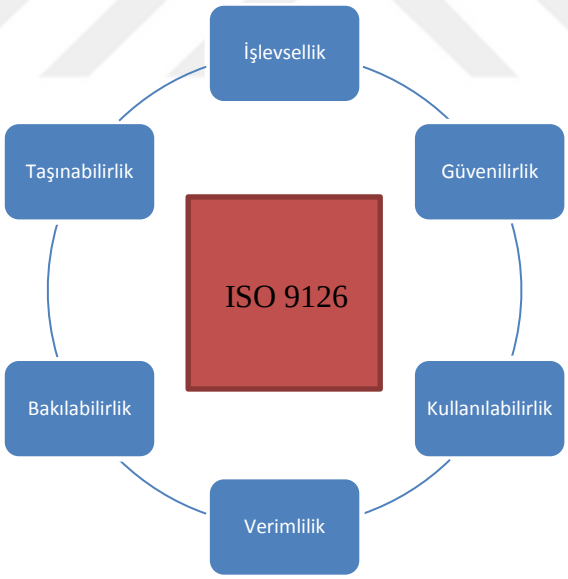
Günümüzde bilgisayarlardaki donanımlar düşük hata oranı ve maliyetler ile üretilebilirken yazılım tarafındaki hata ve maliyet oranları çok daha yüksek seviyelerdedir. Yazılım projelerinin büyüklüğünün artması geliştirme süresi ve bakım masrafları bakımından bir artışa sebep olmuştur (Kitchenham 2010). Günümüzde, geliştirmesi yapılmış birçok yazılım projesi başarısızlıkla sonuçlanıp, yazılım çöplüğünde kendine yer bulmaktadır.

Başarısız yazılımların en çarpıcı örneklerinden biri Amerikan ordusunda yazılım projeleri üzerinde yaptığı bir araştırma sonrasında görülmüştür. Bu araştırma sonucunda yazılım projelerinin %47'sinin insan hayatında hiç yer edinmediği, %29'unun yazılımı talep eden kurum tarafından kabulü yapılmadığı, %19'unun proje başlangıcından kısa bir süre sonra iptal edildiği veya ciddi oranda değiştirilerek tekrar başlatıldığı, %3'ünün ise birkaç ciddi değişiklik yapıldıktan sonra kullanılabildiği tespit edilmiştir. Sadece %2'si teslim alındığı şekliyle kullanılmaya devam edilmektedir (Núñez-Varela et.al.2013).

Bu yüksek lisans tez çalışmasında sık kullanılan yazılım metrikleri, bu metriklerin sonuçlarının değerlendirilip analiz edilmesi ve yazılım ölçüm metrikleri ile ilgili yapılan çalışmalar anlatılmış ve bazı yazılım kütüphanelerine ait metrik sonuçları hesaplanarak yazılım maliyeti tahminlemedeki hata analizi çalışması yapılmıştır. Yapılan tahminleme için kullanılacak veri setinin herkes tarafından erişilebilir açık kaynak kodlu yazılım projeleri olması göz önünde bulundurulmuştur. Bunun sebebi araştırma ve hata oranı sonuçlarının istenildiğinde teyit edilmesi için uygun ortam oluşturabilmektir. Yazılım projeleri GitHub ve Maven Repository kaynaklarından elde edilmiştir. Devamında tüm yazılım projeleri için yazılım metrik sonuçları, Source Monitor uygulaması ile

hesaplanarak elde edilmiştir. Source Monitor yazılım hesaplama aracı aynı şekilde herkes tarafından Campwood Software firmasına ait internet sitesi üzerinden ücretsiz temin edilip birçok programlama dili için desteği olan bir masaüstü uygulamasıdır. Hesaplanan metrik sonuçları üzerinden yapılan veri madenciliği için Rapidminer Studio isimli masaüstü uygulaması kullanılmıştır. Rapidminer yıllık kiralama üzerinden lisanslama yapılan bir uygulamadır, ancak bu çalışmada 1 yıllık öğrenciler için sağlanan ücretsiz lisans kullanılarak tez çalışması tamamlanmıştır.

Bilişim sektöründeki yazılım proje maliyetlerinin bu denli büyümesi sonucu, yazılımda kalite kavramı en çok üzerinde çalışılan konulardan birisi haline gelmiştir. Crosby, genel anlamda kaliteyi "isterlere uyumluluk" olarak tanımlamaktadır (Crosby 1980). Yazılım kalitesini çeşitli sınıflara ayırarak incelemek mümkündür. ISO 9126, kalite kavramının yazılım üzerindeki sınıflandırmasını yapan ve kaliteyi açıklayan bir uluslararası standarttır. Bu standarda göre kalite sınıfları şunlardır (Palıgu, Yağcı ve Öztürk 2013):



Şekil 1.1 ISO 9126 SQM kalite sınıfları

İşlevsellik: Yazılımda kullanıcı ihtiyaçlarını karşılama oranı
Güvenilirlik: Yazılımın doğru çalışabilirliğini muhafaza edebilmesi
Kullanılabilirlik: Kullanım kolaylığı sağlayan yeteneklerin muhafaza edilmesi
Verimlilik: Yazılımın ihtiyaç duyulan performansa sahip olabilmesi
Bakılabilirlik: Değişikliklere adaptasyon, iyileştirme yapılabilirliği
Taşınabilirlik: Farklı çalışma ortamlarına uyum sağlayabilmesi

2. KURAMSAL TEMELLER VE KAYNAK ÖZETLERİ

2.1 Özellik Tabanlı Yöntemler

2.1.1 Code coverage (Kod kapsamı)

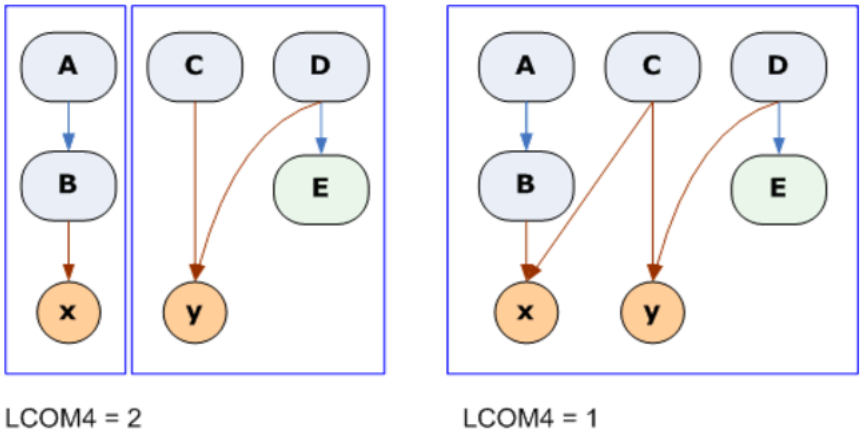
Projedeki kodun, birim testler tarafından ne oranda kapsandığını hesaplamayı sağlayan metriktir. Kodun bakımının sürdürülebilirliği açısından önemlidir. Aşağıdaki formülle ölçülür (Anonymous 2017a).

```
Code Coverage = (Number of lines of code exercised)/(Total Number of lines of code) * 100%
```

Şekil 2.1 Code coverage formülü

2.1.2 Cohesion (Uyumluluk)

Bunge benzerliği, iki varlık arasındaki özellik kümesinin kesişimi olarak açıklamaktadır (Bunge 1981). Bu ölçüm sınıfın sorumlu olduğu işlerin kendi içinde uyumluluğudur. Bu ölçüme göre her sınıfa ait sorumluluk sayısı bir olması uygundur. Hesaplaması LCOM (Lack of Cohesion Methods) ismindeki uyumluluk metriği ile yapılır. Son LCOM metod versiyonu olan LCOM4 güncel olarak kullanılır (Anonymous 2017b).



Şekil 2.2 Cohesion (Uyumluluk) hesaplama örneği

Düşük uyumluluk aşağıdaki problemlere sebep olabilir:

- Sınıf anlaşılırlığı azalır.
- Tekrar kullanım zorlaşır.
- Değişikliklerden etkilenme oranı artar.

Yukarıdaki maddelere baktığımızda doğrudan yazılımın bakımı ile ilişkili olduğunu görmekteyiz.

2.1.3 Coupling (Bağımlılık)

İki nesneden birinin diğerine etki etmesidir. Nesne tabanlı programlamada farklı iki nesne arasında erişim olması kaçınılmazdır, ancak bu ilişkilerde nesneler uygulama (implementation) aşamasında mümkün olduğu kadar birbirinden bağımsız olmalıdır. Sıkı bağlılık (Tightly-Coupled) olduğunda, ilişkili sınıflardan birinde değişikliğe gidersek, diğer bağlantılı sınıfları da etkileme ihtimali yüksektir. Bu da Bakım maliyetleri anlamında bir yük getirecektir. Bu nedenle yazılım mühendisliğinde nesneler arası ilişkinin sıkı olmaması, tam tersine gevşek bağlı (Loosely-Coupled) olması beklenir (Anonymus 2017c).

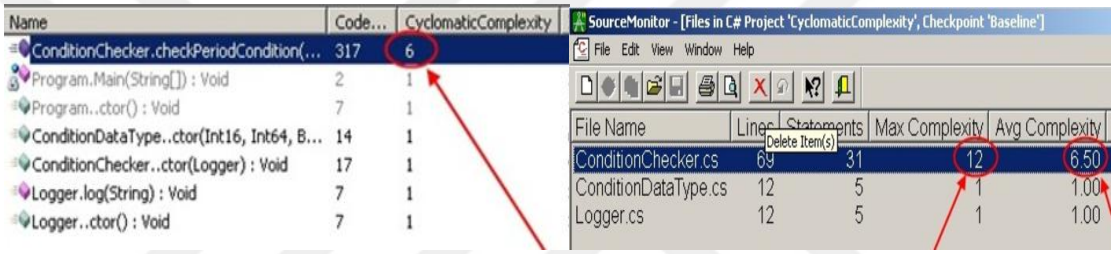
2.1.4 Complexity (Karmaşıklık)

Bu mertriğin en yaygın kullanılan örneği Cyclomatic Complexity'dir. Bir sınıftaki metodda bulunan switch, for, if-else, while gibi karar noktalarının sayılması ile hesaplanır (Anonymous 2018a).

Karmaşıklık metriği ile ilgili dikkat edilmesi gereken konu, bazı yazılım metrik araçları bu hesaplamada "else" komutunu da sayıma dahil ederken bazı araçlar ise if-else bloğunda sadece "if" komutunu hesaplamaktadır (Beratoğlu 2009). Bu durumdan dolayı da A aracında çıkan Cyclomatic Complexity değeri, B aracına göre 2 katı sonuç vermektedir. Java programlama dilini baz alacak olursak, else bloğu uygulamaya ekstra bir yük getirmemektedir. Bu sebeple de if-else karar noktalarının değerinin karmaşıklık değerinin hesaplanmasında 1 olarak eklenmesi daha uygun olacaktır (Anonymous 2017d).

```java
/**
 *
 * @author adem.dilbaz
 */
public class CyclomaticComplexity {

    void receive(Object object) {
        if (object instanceof ObjectTypeA) {
            doSomethingA();
        } else {
            if (object instanceof ObjectTypeB) {
                doSomethingB();
            } else {
                if (object instanceof ObjectTypeC) {
                    doSomethingC();
                } else {
                    if (object instanceof ObjectTypeD) {
                        doSomethingD();
                    } else {
                        // ...
                    }
                }
            }
        }
    }
```

Şekil 2.3 Karmaşıklık hesaplama için örnek Java kodu (Complexity= 4)



Şekil 2.4 Aynı sınıf için iki farklı yazılım aracının karmaşıklık hesaplaması – Reflector ve SourceMonitor

2.1.5 Diğer yazılım metrikleri

Bu yazılım metriklerine ek olarak aşağıdaki ölçüm çeşitleri de popüler olarak kullanılmaktadır (Anonymous 2017e).

- Response For Class (RFC): Bir sınıf içerisinde tanımlanmış bulunan ve kullanılan toplam metod sayısını ifade eder. Kod bakım maliyeti ile bu metrik için hesaplanan değer doğru orantılıdır.
- Weighted Methods For Class (WMC): Bir sınıf içerisinde tanımlanmış toplam metod sayısını ifade eder. Minimum 6 ve maksimum 33 olarak ideal aralık belirlenmiş de olsa Cohesion hesaplaması bu metriğe göre daha önemli kabul edilir.
- Depth of Inheritance Tree (DIT): Miras (Inheritance) kavramına göre bir sınıfa ait kaç temel (parent) sınıf olduğunu ifade eder. Minimum 2 ve maksimum 6

olarak optimum değerler kabul edilir, 6 üzeri olduğunda test edilebilirlik oranı zayıf, 2 altı olduğunda nesnesel programlamanın yeteri kadar kullanılmadığı anlamına gelir (Lanza and Marinescu 2011).

- Number of Method in Class (NOM): Bir sınıfta kullanılan metod sayısıdır. 40 üzeri metod varsa bu sınıfın bölünmesi gerektiği anlamını taşır. Tek başına bir metrik olmaktan çok LCOM (Lack of Cohesion in Methods) metriği ile beraber değerlendirilir.
- Specialization Index (SIX): Koddaki karmaşıklık değeri ve yazılımdaki bakım maliyet sonuçlarını azaltmak için overload edilen fonksiyonların olabilecek minimum sayıya sahip olması gereklidir. (Anonymous 2017f).

Çizelge 2.1 SIX hesaplama formülü

$SIX = \frac{NMO + DIT}{NMI + NMA + NMO} x100$	
Değişken adı	**Açıklama**
DIT	Kalıtım seviyesi
NMA	Kalıtıma eklenen işlem (metod) sayısı
NMI	Inherit edilen operasyon sayısı
NMO	Aşırı yüklenmiş (overloaded) metod sayısı

- Cyclomatic Density: Koddaki if-else, switch, for, while gibi karar noktalarının toplam çalıştırılabilir (executable) koda oranıdır. Bu oran arttıkça kodun performansında azalma beklenebilir.
- Kod Büyüklüğü (Lines of Code): Genellikle bir yazılım ürününün boyutu satır sayısı ile ölçülür.

  SLOC (Source Lines of Code), kodun yorum ve boşluk satırlarından arındırılmış halidir.

  EXEC (Executable Statements), çalıştırılabilir yordam sayısı, yazılım içinde yer alan çalıştırılabilir satır sayısıdır.

- Yorum Oranı (Comment Percentage): Yazılım için hazırlanmış yorum satır sayısının, toplam yorum ve boşluk içermeyen (SLOC) satır sayısına oranıdır. Yorum satırları arttıkça yazılımın anlaşılabilirliği artacaktır, bu da bakım esnasındaki iş yükünü azaltacaktır.

2.1.6 Yazılım metrik araçları

Toplamda 100'den fazla yazılım metrik hesaplama aracı bulunmaktadır. Bu araçların bazıları masaüstü uygulaması olarak çalışırken, bazıları ise kullanılan geliştirme ortamı (Integrated Development Environment) üzerinde eklenti (plugin) olarak çalışmaktadır. Özellikle Eclipse geliştirme ortamı üzerinde birçok eklenti mevcuttur. Yazılım metrik hesaplamaları yapan araçlar kullandıkları metriklere ve destekledikleri programlama dillerine göre ayrılmaktadır. Aşağıdaki tablolarda popüler olarak kullanılan bazı araçlara ait bilgiler yer almaktadır. Bu listeye ek olarak FindBugs, Eclipse Metrics, PMD, Coverlipse, CheckStyle ve SDMetrics araçları verilebilir (Lincke et al 2008).

Çizelge 2.2 Yazılım metrik araçları ve kullanılan metrikler

YAZILIM METRİK ARAÇLARI	**KULLANILAN METRİKLER**								
Araç Adı	CBO	DIT	LCOM-CK	LCOM-HS	NOC	NOM	RFC	TCC	WMC
SourceMonitor	□	□	□		□	□			
Analyst4j	□	□	□		□	□	□		□
Chidamber & Kemmerers	□	□	□		□	□	□		
CCCC	□	□			□	□			
Understand for Java	□	□	□		□	□			
Dependency Finder		□			□	□			
OOMeter	□	□	□		□			□	
Eclipse MP 1.3.6		□		□	□	□			□
VizzAnalyzer	□	□	□		□	□	□	□	□
Eclipse Metrics 3.4			□	□					□
Semmle		□	□	□	□	□	□		

Çizelge 2.3 Yazılım metrik araçları ve desteklenen programlama dilleri

Araç Adı	Hesaplanan Metrikler	Desteklenen Diller
McCabelQ	Cyclomatic complexity, Halstead, LOC gibi metrikleri içeren 43 metrik	Java, C++, Ada, C, C#, .NET, COBOL, FORTRON, JSP, Perl, PL1, VB, ASM86
CMT++	Cyclomatic complexity, Halstead metrikleri, LOC	C, C++
CMTJava	Cyclomatic complexity, Halstead metrikleri, LOC	Java
RSM	LOC, method ve sınıf sayılarını içeren yaklaşık 100 metrik. Sınıf ve namespace hesaplamaları yapan metrikler	C#, Java, C, C++
JMetric	Cyclomatic complexity, LDC, NOC, NOP	Java
Essential Metrics	Cyclomatic complexity, satır sayısı metrikleri, halstead metrikleri, nesne tabanlı metrikler	Java, C, C+++
DXCore	Cyclomatic complexity, LOC, maintenance complexity	C#, ASP.NET, Visual basic, C++, XML, HTML, XAML,
SourceMonitor	NOF, LOC, NOS, % of Branches, Number of Calls, Code coverage, Number of Classes, Number of methods, Max complexity, max depth, average depth, average complexity	C++, C, C#, VB.NET, Java, Delphi, Visual Basic (VB6) or HTML

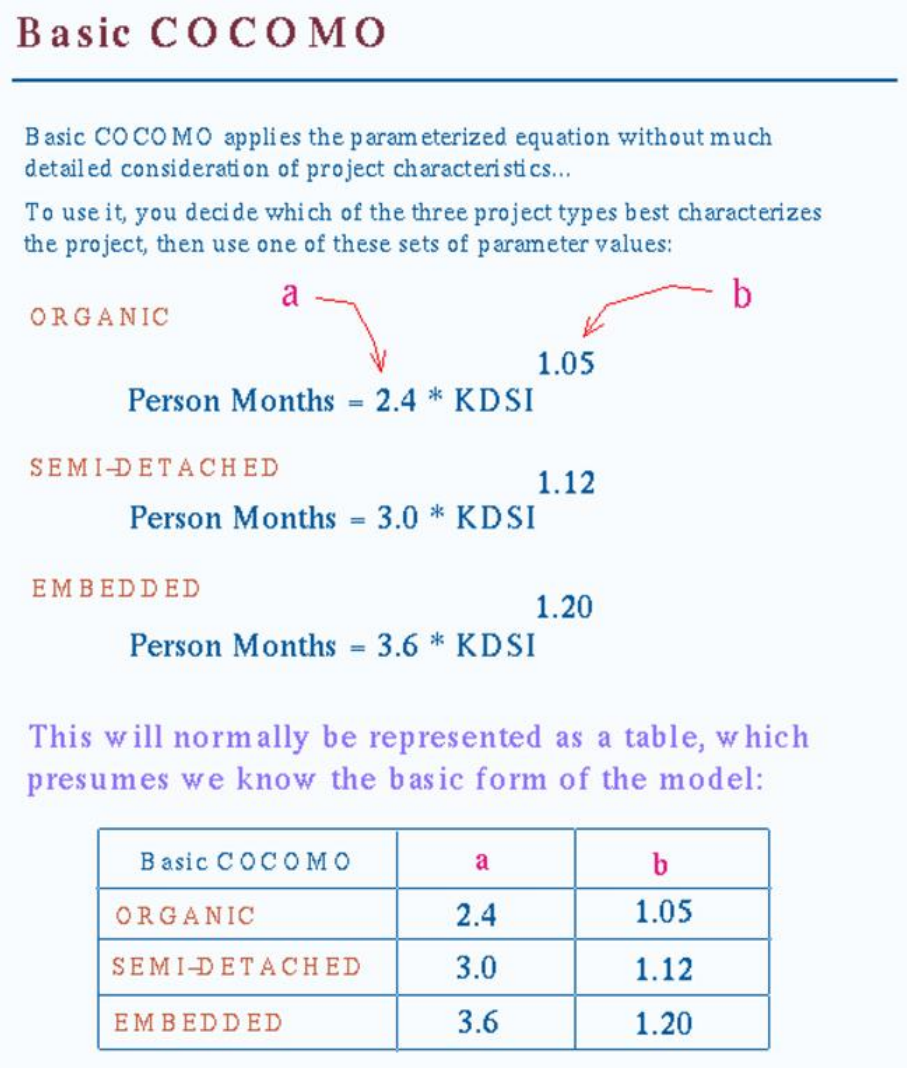
Bu ölçüm araçlarından birini tanıtacak olursak, FindBugs, Maryland Üniversitesi menşeili açık kaynak kodlu bir statik kod analiz aracıdır (Radjenović et al 2013). Java dil desteği bulunmaktadır ve kod analizi yaparak popüler yazılım hatalarını ve yazılım tasarımındaki yanlışları makul kabul edilebilecek bir süre içerisinde bulabilmektedir. Bu araç JBoss, Netbeans, Java gibi günümüzde popüler olan birçok programın geliştirme aşamasında aktif olarak kullanılmaktadır. Örneğin, Netbeans Geliştirme Ortamı uygulamasında 6.0-8m versiyonu FindBugs aracı ile analiz edilerek 189 hata tespit edilmiştir (Anonymous 2017g).

Metric	Total	Mean	Std. Dev.	Maximum
⊞ Number of Overridden Methods (avg/max per type)	11	2,2	2,926	8
⊞ Number of Attributes (avg/max per type)	9	1,8	2,713	7
⊞ Number of Children (avg/max per type)	3	0,6	1,2	3
⊞ Number of Classes (avg/max per packageFragment)	5	5	0	5
⊞ Method Lines of Code (avg/max per method)	206	6,438	10,105	40
⊟ Number of Methods (avg/max per type)	32	6,4	6,681	19
⊟ src	32	6,4	6,681	19
⊟ (default package)	32	6,4	6,681	19
⊞ SortItem.java	19	19	0	19
⊞ SortAlgorithm.java	7	7	0	7
⊞ QSortAlgorithm.java	4	4	0	4
⊞ BidirBubbleSortAlgorithm.java	1	1	0	1
⊞ BubbleSortAlgorithm.java	1	1	0	1
⊞ Nested Block Depth (avg/max per method)		1,688	0,982	4
⊞ Depth of Inheritance Tree (avg/max per type)		2,4	1,356	5
⊞ Number of Packages	1			
⊟ McCabe Cyclomatic Complexity (avg/max per method)		2,438	2,703	12
⊟ src		2,438	2,703	12
⊟ (default package)		2,438	2,703	12
⊞ QSortAlgorithm.java		3,75	4,763	12
⊞ BidirBubbleSortAlgorithm.java		10	0	10
⊞ SortItem.java		1,947	1,669	8
⊞ BubbleSortAlgorithm.java		6	0	6
⊞ SortAlgorithm.java		1,429	0,495	2
⊞ Total Lines of Code	300			
⊞ Instability (avg/max per packageFragment)		1	0	1
⊞ Number of Parameters (avg/max per method)		0,812	0,845	3
⊞ Lack of Cohesion of Methods (avg/max per type)		0,255	0,324	0,776
⊞ Normalized Distance (avg/max per packageFragment)		0	0	0
⊞ Specialization Index (avg/max per type)		1,321	0,89	2,105
⊞ Weighted methods per Class (avg/max per type)	78	15,6	11,074	37

Şekil 2.5 FindBugs aracı çıktısı

2.2 Yazılım Metrikleri Kullanılan Çalışmalar

Bu yüksek lisans tez çalışması için gelecekte yazılım maliyet tahminlemesi yapılması planlanmaktadır. Yazılım maliyet tahmini ile ilgili bazı akademik çalışmalar yapılmıştır. Bunlardan en çok bilineni COCOMO'dur. (Constructive Cost Model). COCOMO, 1981 yılında Boehm tarafından tarafından yayımlanmıştır. Yazılım projelerinde yazılan satır sayısını, yazılım geliştirme ekibinin ve projenin kaç adam/gün'e mal olacağını tahmin eden bir modeldir (Boehm et al 1995).



Şekil 2.6 COCOMO modeli (Anandhi ve Chezian 2013)

COCOMO-81 versiyonundan COCOMO 2'ye dek bir çok sürümü mevcuttur. Basic COCOMO, projede geliştirme için gerekli olan süreyi ve harcanan eforu ve proje büyüklüğünün sonuç değeri olarak kod satır sayısı bakımından hesaplar. Hesaplama formülleri aşağıdaki gibidir (Rajper and Shaikh 2016):

- Efor = A_{Basic} x (LOC)^B_{Basic} (Adam x Ay biriminde)
- Geliştirme Süresi = C_{Basic} x (LOC)^D_{Basic} (Ay biriminde)
- İnsan Maliyeti = Efor / Geliştirme Süresi (Adam biriminde)

Farklı proje türlerine göre yukarıdaki formüllerin uygulanması için A_{Basic},B_{Basic},C_{Basic}, D_{Basic} parametrelerinin değerleri Şekil 2.4'deki gibi olmalıdır.

Çizelge 2.4 COCOMO formül parametreleri (proje türlerine göre)

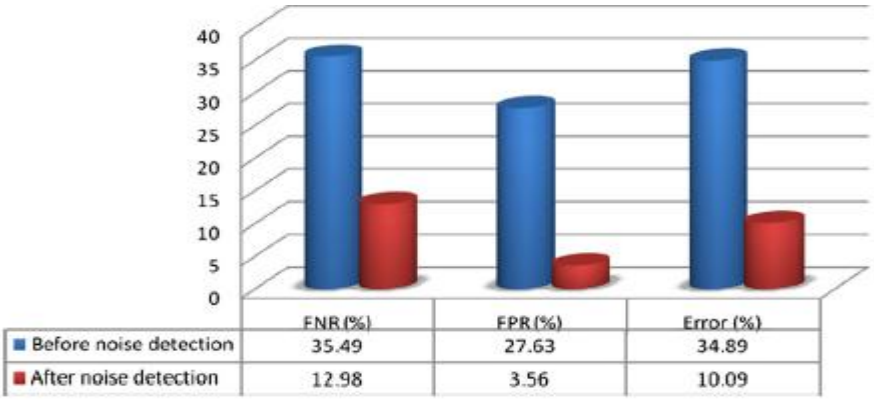
	A_{Basic}	B_{Basic}	C_{Basic}	D_{Basic}
Organic	2.4	1,05	2i5	0,38
Semidetached	3	1,12	2,5	0,35
Embedded	3,6	1,2	2,5	0,32

Başka bir çalışmada ise, yazılım metriklerini ve ROC (Receiver Operation Characteristics) eğrilerini analiz ederek gereksiz verilerin tespit edilmesi konusunda bir algoritma oluşturulmuştur. Yazılım metrikleri eşik değerleri (thresholds) baz alınarak 5 NASA veri seti üzerinde veri madenciliği yapılarak bazı sonuçlar elde edilmiştir. Kullanılan formüller ve parametrelerdeki değişiklik Şekil 4-3'teki gibidir (Catal et al 2011).

$$FPR = \frac{FP}{FP + TN}$$

$$FNR = \frac{FN}{FN + TP}$$

$$Error = \frac{FN + FP}{TP + FP + FN + TN}$$



Şekil 2.7 Gereksiz verilerin tespiti sonucu bazı parametrelerdeki değişimler

Yazılım metrikleri kullanılarak yapılan diğer bir çalışmada ise, nesne tabanlı dizayn metrikleri kullanılarak, bu metriklerin yazılımın kod büyüklüğüne etkisi incelenmiştir. MOOD Suite, QMOOD Suite, Martin Suite, CK Suite ve diğer bazı metrik kümelerindeki yazılım metrikleri, aralarında tekrar eden (duplicate) metrik olmayacak şekilde bir araya getirilmiştir. Bu metriklerin açıklama ve hesaplanma süreçleri incelenerek aynı ya da yakın ilişkili metrikler ayıklanıp yeni bir alt metrik kümesi oluşturulmuş ve böylece çalışmada kullanılacak metrikler belirlenmiştir. 252 paket üzerinde metriklerin hesaplanması için Eclipse geliştirme ortamı üzerindeki Metric eklentisi kullanılmıştır ve metrikler arası ilişkiler gösterilerek kod büyüklüğünün tahmini için bir model oluşturulmuştur (Tanriover and Eryigit 2015).

$$MLOC=\beta_0+\beta_1 NOC+\beta_2 NOM+\beta_3 NOF+\beta_4 DIT+\beta_5 CA+\beta_6 CE+\beta_7 LCOM$$

(DENKLEM) Kullanılan Regresyon Modeli

Bu çalışmanın çıktıları olarak MLOC, NOC, NOM, NOF metrikleri arasında yüksek oranda benzerlik gözlemlenmiştir. Aynı zamanda DIT metriğinin ise kod satır sayısını (LOC) negatif yönde etkilediği tespit edilmiştir. Kullanılan metrikler yazılım ölçüm araçlarıyla otomatik olarak hesaplayabileceği için, bu regresyon modeli ile kod satır sayısı tahmini yapılabilir. Aynı zamanda bir yazılımın farklı versiyonları için yazılım kalite kriterlerinin hesaplanarak kıyaslanması ve takibini yapmak mümkündür. Gelecekte yapılması düşünülen çalışma olarak da dizayn metriklere ek olarak UML model metriklerinin de eklenerek yazılım boyut tahmini yapılması planlanmaktadır.

3. GELİŞTİRİLEN YÖNTEM

Bu bölümde; yazılım metrikleri çalışmalarının genel yapısı, tez kapsamında geliştirilen algoritma ve kullanılan yöntemler detaylı bir biçimde anlatılmıştır.

3.1 Yazılım Metriklerinde Temel Konsept

Yazılım metrik türleri 4 farklı başlık altında sınıflandırılabilir.

Token Metrikleri

Andaç (token), kaynak koddaki en basit komut olarak ifade edilebilir. Buna örnek olarak class, method, while, for, foreach, if gibi anahtar kelimeler verilebilir. Token metrik türünde sonuçlar bir kaynak kodunun tamamı veya belirli bir bölümü için bu tür andaçların sayılarak hesaplanması süreci vardır. Farklı kaynak kodlar için bu metrikler hesaplandığında sayım sonunda ortaya çıkan istatistiksel veriler kıyaslanabilir (Anonymous 2018b).

Denetim Akış Metrikleri

Bir yazılım geliştirme sistemi içerisindeki her birim kontrol edilerek denetim akışı ölçülür. Çalışılan birimin denetim yapısının karmaşıklık değeri çeşitli veri görselleştirme örnekleriyle ifade edilir.

Bileşik Metrikler

Token ve denetim akış metriklerinin birlikte hesaplanması ile elde edilir.

Sistem Metrikleri

Sistem metrikleri diğer metriklerin aksine kaynak kodun özelliklerini ölçmek yerine sistemin tasarımı ile ilgili ölçümlemeler yaparlar. Bunlar süreç, ürün niteliği, üretkenlik, bakılabilirlik boyut, zamanlama gibi büyük ölçekli tasarım nitelikleridir. Bu metrikler, daha kaynak kod yazılmaya başlamadan gereksinimlerin belirlenmesi evresinde hesaplanmaya başlayacağı için projenin ilerleyişini tahminlemede ciddi kazanç sağlarlar (Hill 2011).

Sistem metrikleri büyük ölçekli yazılım projelerinde yüksek seviyede önem arz eder. Yaygın olarak kullanılan sistem metrikleri şunlardır:

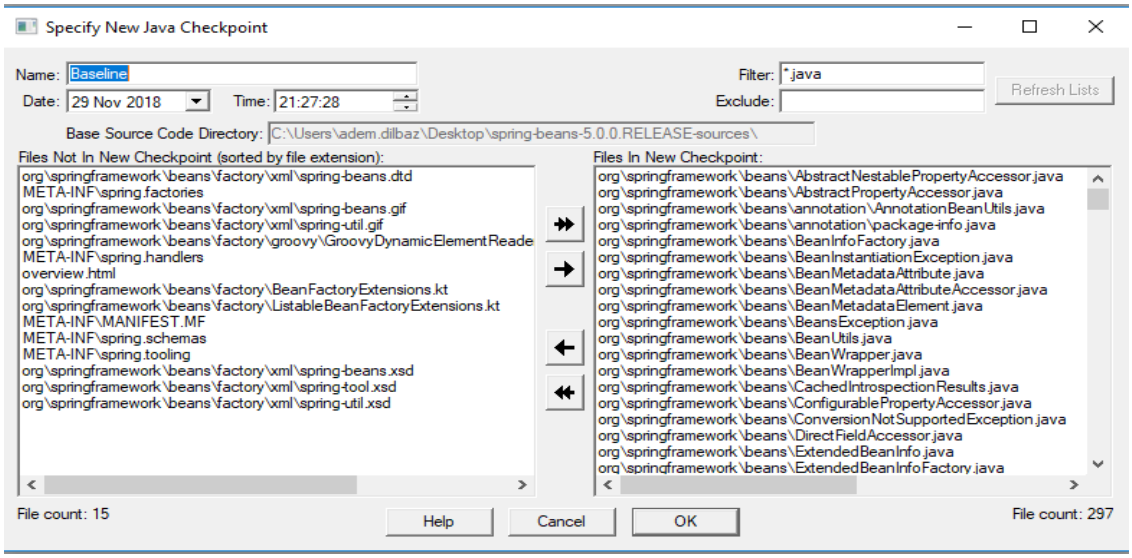
- Süreç metrikleri: Bir yazılım geliştirme sürecinde kullanılan tekniklerin, araçların, ekibin, altyapının özelliklerinden oluşur. Bu metriklerin yardımıyla bu süreçteki geliştirme araçlarının etkinliği, kod yazan kişilerin teknik seviyesi, altyapının yeterliliği ölçümlenir (Anonymous 2018c).
- Boyut metrikleri: Projenin gereksinimleri belirleme aşamasında hedeflenen isterlerle, mevcut durumdaki isterlerin ne derece örtüştüğünü hesaplayıp durum değerlendirmesi yapılması sağlanır (Anonymous 2018d).
- Zamanlama metrikleri: Projenin başlangıçta belirlenen takvime uygun ilerleyip ilerlemediği sonucuna ulaşır. Tahmin edilen toplam süre ve mevcut duruma kadarki geçen süre kıyaslanır.
- Maliyet ve kaynak metrikleri: Proje başlangıcında tahmin edilen adam/gün sayısı mevcut duruma kadar harcanan adam/gün sayısı ile oranlanır. Bu hesaplamada birim olarak para da kullanılabilir.
- Ürün nitelik metrikleri: Karşılaşılan ve kayıt altına alınan yazılım kusurlarının türü, önem seviyesi ve zaman parametreleri göz önünde bulundurularak ürünün satılmadan önceki kusur sayısı ile hesaplanır.
- Bakım ve okunabilirlik metrikleri: Koddaki okunabilirliği ölçen metriklerdir. Örneğin değişkenlerin içinde bulunan kod satır sayısı 50'den fazla olmamalıdır, açıklama satır sayısı çalıştırılabilir yordam sayısının 1/3'inden fazla olmalıdır gibi. Bu metrikler hesaplanarak farklı yazılımlar arasında kıyaslama yapılır.
- Üretkenlik metrikleri: Projedeki her aşamada ölçülebilecek metriklerdir. Bunlardan bazıları, kod hatalarının adam/gün değerine oranı veya boyut metriklerinin harcanan adam/gün maliyetine oranı gibi.

3.2 Yazılım Metrikleri Uygulama Aşamaları ve Kullanılan Yöntemler

3.2.1 Source Monitor

Bu çalışmada yazılım projelerini ölçümlemek amacıyla SourceMonitor v3.5 aracı kullanılmıştır. SourceMonitor, Campwood Software tarafından sağlanan açık kaynak ve ücretsiz bir metrik hesaplama aracıdır. C++ dilinde yazılmıştır ancak Java dahil birçok yazılım projesi için metrikler hesaplayabilmektedir. Online bir araç değil, masaüstü uygulaması olarak çalışmaktadır (Voulgaropoulou et al 2011).

Ölçümleme yapılacak projeye ait jar dosyası dizine çıkartıldıktan sonra ilgili dizin yolu uygulamaya verilecektir. Bu yüzden encrypt ya da obfuscate edilmiş Java kütüphaneleri için bu araç kullanılamayacaktır, kaynak kodunun bulunduğu dizine ihtiyaç vardır. Dizin yolu verildikten sonra bir checkpoint ismi belirlenmekte ve sonrasında aşağıdaki gibi dahil edilen veya hariç tutulan dosya bilgisi listelenmektedir. Bu ekranda hesaplamaya dahil edilmesini istemediğimiz sayfaları çıkarabiliriz ancak çalışmada tüm proje için hesaplama yapılacağından listelerde bir değişiklik yapılmamıştır.



Şekil 3.1 Source Monitor aracı Java checkpoint oluşturma

Bu adım geçildiğinde ise ölçüm sonuçlarının listelendiği ekran ile karşılaşılmaktadır. SourceMonitor dosya sayısı, kod satır sayısı (LOC), ifade, kod dalları, metod çağırılma

sayısı, yorum satır oranı, sınıf sayısı, metod sayısının sınıf sayısına oranı, ortalama ifadenin metod sayısına oranı, maksimum karmaşıklık, maksimum derinlik, ortalama derinlik ve ortalama karmaşıklık gibi yazılım metriklerinin ölçümlerini yapmaktadır.

Çizelge 3.1 Source Monitor aracı çıktısı

Checkpoint	**Files**	**Lines**	**Statements**
Spring-core	297	50967	16921
%Branches	**Average Statements/ Method**	**Max Depth**	**Classes**
16.5	4.15	9	338
Methods / Class	**Calls**	**Maximum Complexity**	**% Comments**
6.94	7754	58	40.7
Avg Depth	**Avg Complexity**		
2.14	2.53		

Veri setinde farklı versiyonları bulunan 103 farklı Java uygulaması SourceMonitor aracına girdi olarak verilmiş ve tüm sonuçlar ortak bir Excel dosyasına aktarılmıştır.

3.3 Yazılım Projeleri

Bu çalışmada Github ve Maven Repository üzerinden 103 farklı Java kütüphanesi temin edilerek bu projelerin farklı major versiyonları veri seti olarak kullanılmıştır. Spring Framework, Jetty, Hibernate Framework, Maven, OpenJPA, Apache Tomcat ve Finagle uygulamalarına ait ait başlıca Java kütüphaneleri veri setine dahil edilmiş ve bu projelerin kod satır sayısı 1500 üzerinde olması göz önünde bulundurulmuştur. Bu projeler birden fazla farklı versiyonları ile birlikte Source Monitor aracına girdi olarak verilmiş ve projelerin ilgili versiyonları için çeşitli yazılım metrikleri hesaplanmıştır.
Ek-1'de bu çalışmada kullanılan Java dilinde yazılmış projelere ait kütüphane isimleri yer almaktadır.

3.4 Hesaplanan Metrikler

Toplamda 13 farklı yazılım metriği SourceMonitor uygulaması ile her bir proje için hesaplanmıştır. Hesaplanan metrikler aşağıdaki gibidir.

Çizelge 3.2 Çalışmada hesaplanan metrik çeşitleri

Files	Statements	Rate of Branches	Calls
Rate of Comments	Classes	Methods/Classes	Average Statements per Method
Maximum Level of Complexity	Average Depth	Method	Average Complexity
Line of Code (LOC)			

- Files: Bir projedeki .java uzantısına sahip dosya sayısı
- Statements: Projede yer alam yordam sayısıdır.
- % Branches: Yazılan birim testlerin ilgili dalları kapsama oranı. 100% olduğunda tüm kod için test işlemleri yapıldığı anlamı taşır. Örneğin bir if-else koşulunda her iki koşulun da en az 1 defa çalıştırılması, böylece testin her iki durumu da kapsaması amaçlanır. Formülü aşağıdaki gibidir:

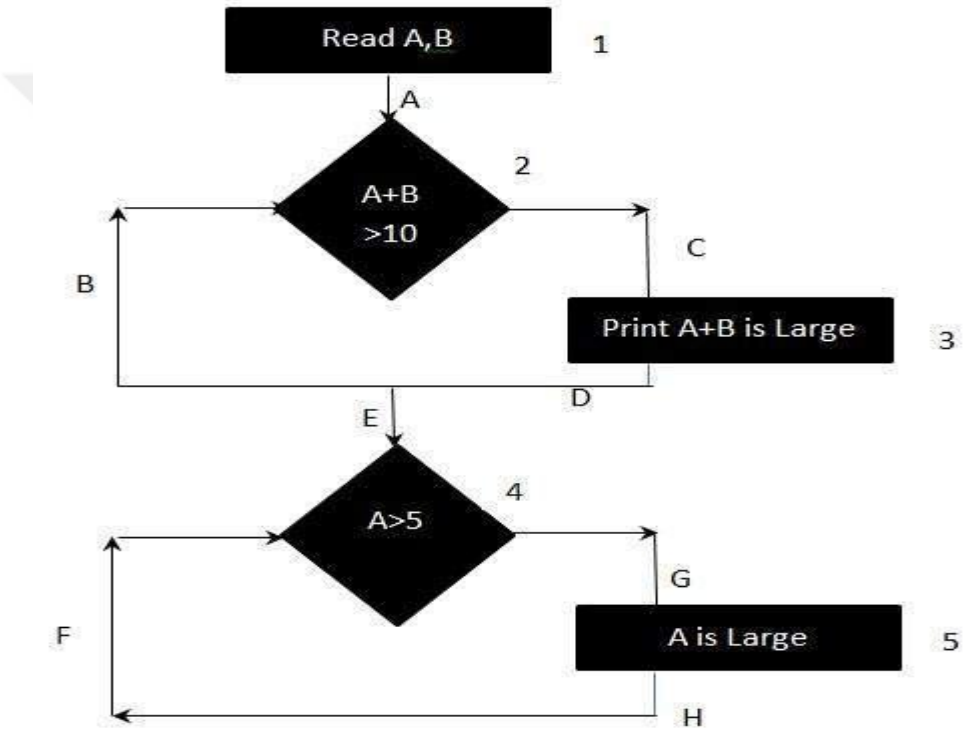
Kapsama Oranı = (Test edilen karar mekanizmalarının sayısı / Toplam karar mekanizmalarının sayısı) x 100 %

Aşağıdaki kod bloğunda 2 farklı koşul bulunmaktadır.

```
public void kıyaslama(Integer A, Integer B){
     if((A+B) > 10){
        System.out.print(“A ve B toplamı büyüktür.”)
     }
     if(A > 5){
        System.out.print(“A büyüktür.”)
     }
}
```

Buradaki yapıyı aşağıdaki gibi akış diyagramına dönüştürebiliriz. Diyagrama göre tüm karar mekanizmalarını aynı anda kapsanmasını sağlayacak tek bir yol yoktur. Amaç tüm true/false karar mekanizmalarının kapsanmasıdır. Takip edilebilecek yollar aşağıdaki gibidir:

Yol 1: 1A | 2C | 3D | E | 4G | 5H
Yol 2: 1A | 2B | E | 4F
Bu durumda kapsama oranı değeri 2 olarak belirlenmektedir.



Şekil 3.2 % Branches diyagram gösterimi

- Number of Calls: Projedeki metodların toplam çağırılma sayısıdır.
- % Comments: Projedeki yorum sayısının kod satır sayısına oranıdır.
- Classes: Projedeki toplam sınıf sayıdır.
- Methods/Class: Sınıflardaki ortalama metod sayısıdır.
- Average Statements/Method: Metodlardaki ortalama yordam sayısıdır.
- Maximum Cyclomatic Complexity: Projedeki kodun ne kadar karmaşık olduğu hakkında bilgi verir. Bu hesaplamaya koddaki if,else,while,switch gibi karar ve

döngü mekanizmaları dahil edilir. Her bir metod için complexity değerine göre o metodun bakım maliyeti ya da riski aşağıdaki gibi kabul edilir (Eisty et al 2018).

Çizelge 3.3 Karmaşıklık değeri – Risk değeri tablosu

Cyclomatic Complexity	**Risk**
1-10 arası	Düşük Seviyeli Risk, Basit
11-20 arası	Orta Seviyeli Risk, Daha Karmaşık
21-50 arası	Yüksek Seviyeli Risk, Karmaşık
50'den fazla	Çok Yüksek Seviyeli Risk, Test Edilemeyen

Cyclomatic Complexity formülü şu şekildedir: E- N + 2 * P

E: Akış şemasındaki kenar sayısı (edge)

N: Akış şemasındaki karar düğümü sayısı (node)

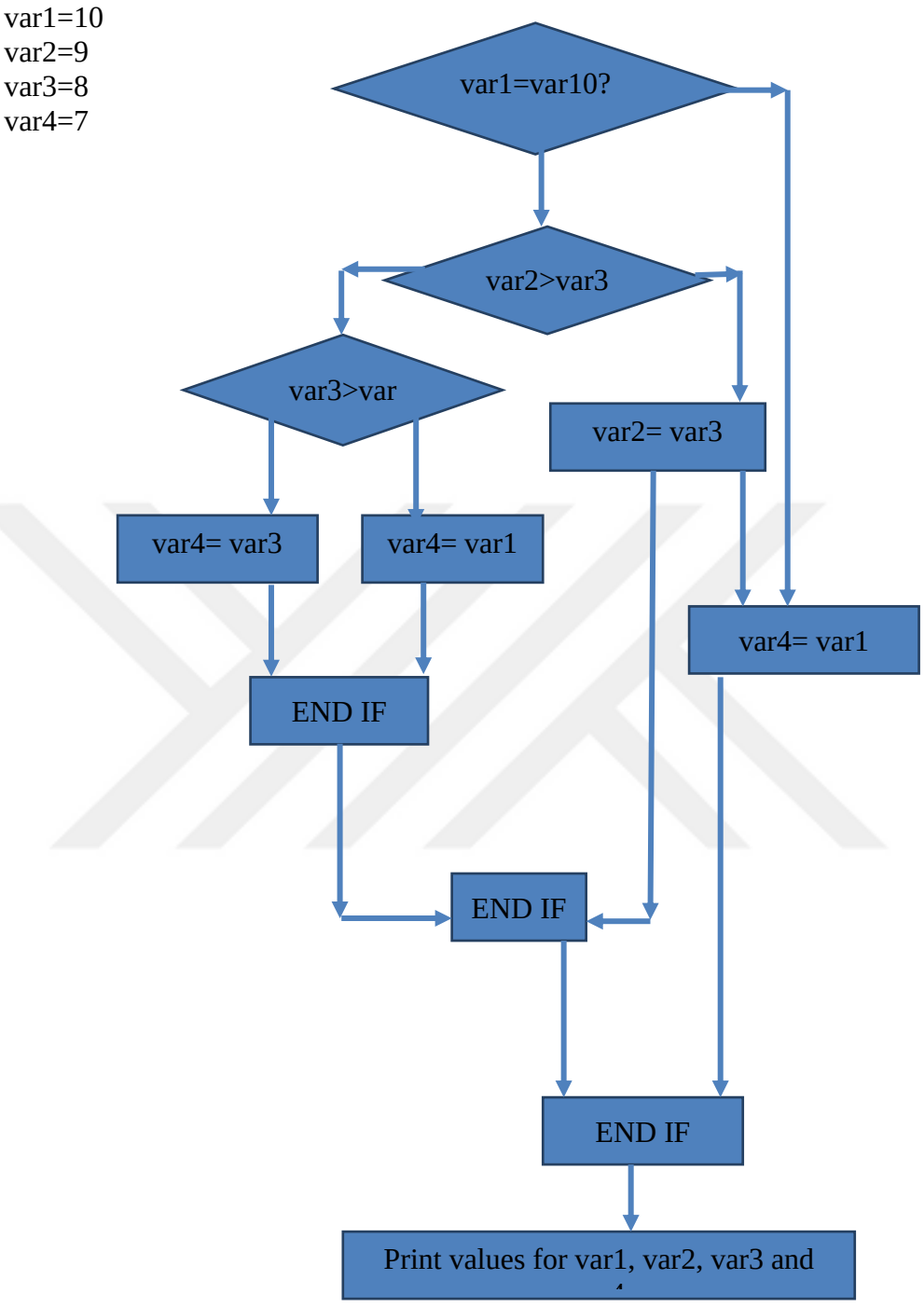
P: Aşı şemasındaki çıkış yoluna sahip karar düğümü sayısı

Bu bilgiler ışığında aşağıdaki demo Java kodu incelendiğinde 3 if ve 3 else koşulu olduğu görülmektedir.

```java
public class CyclomaticComplexityDemo {

	public static void main(String[] args) {

		// TODO Auto-generated method stub

		int var1 = 10;
		int var2 = 9;
		int var3 = 8;
		int var4 = 7;

		if (var1 == 10){

			if(var2 > var3){

				var2 = var3;

			}

			else{

				if (var3 > var4){

					var3 = var4;

				}

				else{

					var4 = var1;

				}

			}

		}

		else{

			var1=var4;

		}

		System.out.println("Printing value for var1, var2,
var3, and var4"+var1+" "+var2+" "+var3+" "+var4);

	}

}
```

Şekil 3.3 Karmaşıklık metriği hesaplama – Java kodu

Bu kodun akış diyagramını çizdiğimizde aşağıdaki gibi bir grafik elde edilmektedir.

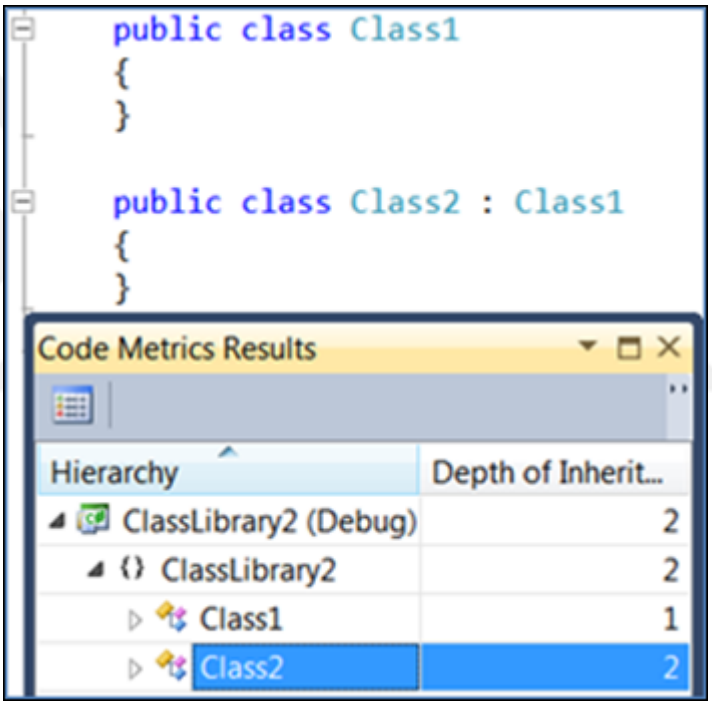


Şekil 3.4 Karmaşıklık metriği için akış diyagramı

Akış grafiğine göre koddaki kenar sayısı 11 (E), karar düğümü sayısı 11 (N), çıkış noktasına sahip karar düğümü sayısı 1 olarak görülmektedir. Cyclomatic complexity değerini hesaplamak için formülü uyguladığımızda 11 – 11 + 2 * 1 = 2 değerine ulaşılmaktadır. Bu sonuçla ilgili kod bloğu için az riske sahip denebilir. Bu metrik

yardımıyla projedeki riskler ve risklerle ilişkili olarak bakım maliyetleri hakkında fikir sahibi olunabilir, proje bütçeleri bu bilgiler ışığında planlanabilir.

- Average Depth (DIT): İlgili sınıfın kalıtım ağacının başlangıcına olan uzaklığı olarak ölçülür. Derinlik (Depth Inheritance Tree) ağacı ne kadar fazla olursa tasarım karmaşıklı o kadar yüksektir. Örneğin aşağıdaki sınıflardan Class1 sınıfı hiçbir sınıftan türemediği için DIT değeri sıfırdır. Class2 sınıfı ise Class1 sınıfından türediği için DIT değeri 1 olarak hesaplanır. Derinlik arttıkça koddaki hata olasılığı ve bakım maliyetleri artmaktadır.
-



Şekil 3.5 Kalıtım ağacı derinlik hesaplama için örnek sınıflar

- Average Cyclomatic Complexity: Bölüm 3.4.9'da Cyclomatic Complexity metriğinin nasıl ölçüldüğünü açıklamıştık. Bu metrikte de bu değerlerin tüm yazılım Projesi için ortalama değeri hesaplanır.
- Number of Method: Bir yazılım projesindeki toplam metod sayısıdır.
- Line of Code: Bir yazılım projesindeki toplam çalıştırılabilir satır sayısıdır. Bu çalışmada satır sayısı 1000'den az olan projeler veri setine dahil edilmemiş, kod

satır sayısı metriği çıktı olarak kabul edilerek tahminleme yapılması amaçlanmıştır.

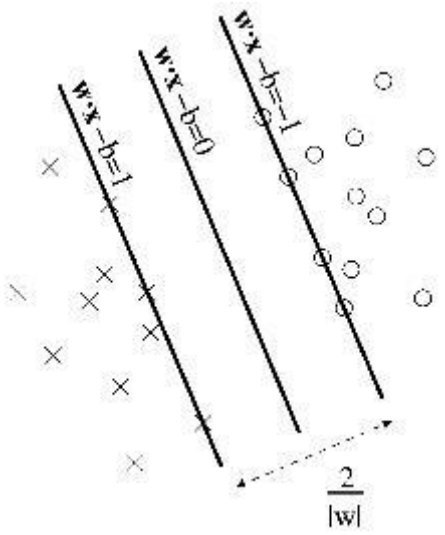
3.5 Metrik Sonuçları

Farklı versiyonlara sahip Java projelerinin SourceMonitor uygulaması ile 14 farklı yazılım metriği hesaplanmıştır. Toplamda 103 farklı Java projesinin 250 farklı major versiyonu hesaplamaya dahil edilmiştir. Sonuçların kendi içerisinde aritmetik ortalaması alınmış olup Ek 2'den tüm sonuçlar görüntülenebilir.

3.6 Makine Öğrenmesi Uygulamaları

3.6.1 Destek vektör makineleri (Support vector machines)

Değişkenler veya gruplar arasındaki ilişkinin tam olarak bilinmediği veri kümelerinde sınıflandırma yapmak amacıyla kullanılan bir yöntemdir. 2 farklı veri seti için 2 boyutlu düzlemde, bu veri setlerini ayıracak en uygun a.x + b doğrusunu çizmeyi sağlar.



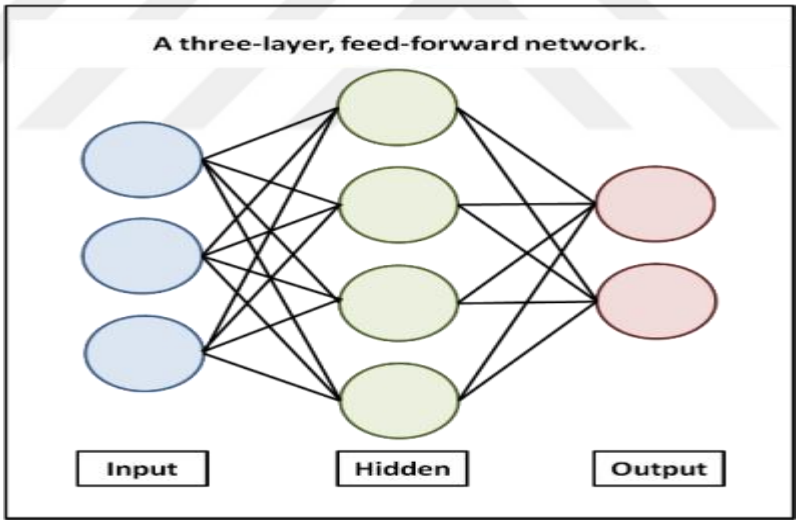
Şekil 3.6 DVM hesaplama doğruları

2 veri setinin sınırlarını belirleyen birer doğru çizildiğinde bu doğrulara en uzak doğru DVM için en uygun doğrudur. Tolerans, 2 farklı sınır doğrusu arasında kalan uzunluğu ifade eder.

Destek vektör makineleri sınıflandırmada yüksek doğruluk sağlamasına rağmen olasılıksal tahminler üretememektedir.

3.6.2 Yapay sinir ağları (Artificial neural network)

Yapay sinir ağları, beynimizdeki öğrenim yapısının matematiksel modellemelerle ortaya konduğu bir makine öğrenme algoritmasıdır. Biyolojik olarak nöronlar gövde, dendrit ve akson olarak 3 ana kısımdan oluşur. Nöronlar sinapslar aracılığıyla diğer nöronlardan gelen uyarılara karşı tepki verirler. Yapay nöron ise bir ya da birkaç girdi alıp buna göre çıktı üretecek şekilde oluşturulmuştur.



Şekil 3.7 YSA katmanları

Yapay sinir ağlarında 5 element vardır:

Girdiler: Ağda kullanılacak veri seti tarafından belirlenen dış taraftan ağa verilen bilgiler

Ağırlıklar: Bir yapay hücreye gelen input değerinin etkisini ifade eder. Bir ağırlığın büyük olması önemli, küçük olması önemsiz olduğu anlamı taşımaz. Sıfır ağırlığı da en önemli ağırlık olabilir.

Birleştirme fonksiyonu: Hücreye gelen girdiyi hesaplayan fonksiyondur. Burada birden fazla fonksiyon yaygın olarak kullanımaktadır. Her gelen bilgi ağırlık değeri ile çarpılır ve girdi hesaplanır.

Çizelge 3.4 Birleştirme fonksiyonları

Toplam Net=$\sum_{i=1}^{N} X_i * W_i$	Weight değerleri girdi değerleri ile çarpıldıktan sonra sonuçlar birbirleriyle toplanıp Net sonuç elde edilir.
Çarpım Net=$\prod_{i=1}^{N} X_i * W_i$	Weight değerleri girdi değerleri ile çarpıldıktan sonra sonuçlar birbirleriyle çarpılıp sonuç elde edilir (Net).
Maksimum Net=Max($X_i * W_i$)	n tane input değeri içerisinden weight değerleri girdilerle çarpılıp en büyük olanı Net girdi şeklinde belirlenir.
Minimum Net=Min($X_i * W_i$)	n tane input değeri içerisinden weight değerleri girdilerle çarpılıp en küçük olanı Net girdi şeklinde belirlenir.
Çoğunluk Net=$\sum_{i=1}^{N} Sgn(X_i * W_i)$	n tane girdi içinden ağırlıklar ve girdiler çarpılıp pozitif ve negatif olanların toplamı hesaplanır. Hücre net girdisi büyük sayıdır.
Kümülatif Toplam Net=Net(eski)+$\sum_{i=1}^{N} X_i * W_i$	Hücredeki değerler ağırlıklı olarak toplandıktan sonra hücreye önceden gelmiş bilgilere yeni hesaplanmış input değerleri eklenip bu hücredeki girdi (input) değeri hesaplanmış olur.

Aktivasyon fonksiyonu: Yapay hücreye gelen girdi bu fonksiyonda uygulanarak elde edilecek çıktının hesaplanması sağlanır. Burada da yine birden fazla fonksiyon kullanılabilir, çoğunlukla doğrusal olmayan fonksiyonlar kullanılır.

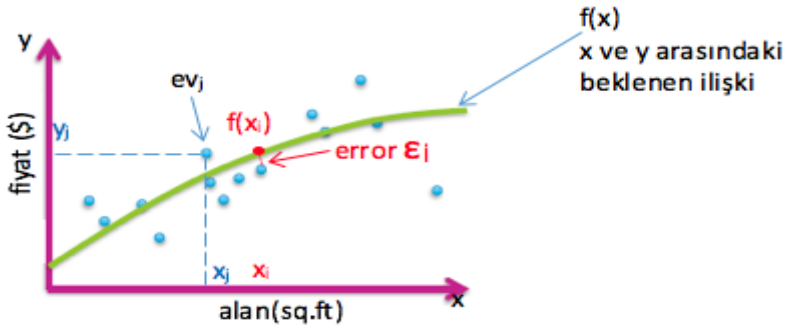
Çizelge 3.5 Aktivasyon fonksiyonlarına ait formül ve açıklamalar

Linear Aktivasyon Fonksiyonu	F(Net)=A*NET (A sabit bir sayı)	Doğrusal problemlerin çözümünde bu fonksiyon kullanılabilir. Toplama fonksiyonunda elde edilen değer sabit bir katsayı ile çarpıldıktan sonra hücrenin sonuç değeri hesaplanmış olur.
Adım (Step) Aktivasyon Fonksiyonu	F(Net)=$\begin{cases} 1 \; if \; Net > Eşik \; Değer \\ 0 \; if \; Net \leq Eşik \; Değer \end{cases}$	Net girdi değeri, belirlenmiş eşik sayıdan fazla ise hücre 1, az ise 0 değerini almış olur.
Sigmoid Aktivasyon Fonksiyonu	F(Net)=$\frac{1}{1+e^{-Net}}$	Bu fonksiyon türevi alınabilen ve sürekli bir fonksiyon olarak düşünülür. Yapay sinir ağı kullanılmış uygulamalarda doğrusal bir fonksiyon olmadığı için en popüler fonksiyon olarak kabul edilir. Her girdi için 0 ve 1 aralığında değer verilir.
Tanjant Hiperbolik Aktivasyon Fonksiyonu	F(Net)= $\frac{e^{+Net}+e^{-Net}}{e^{+Net}-e^{-Net}}$	Sigmoid fonksiyonu ile benzer özelliklere sahipir. Sigmoidde çıktılar 0 ve 1 aralığında değişebilen değerler alırken bu fonksiyonda çıktılar -1 ve 1 aralığındadır.
Eşik Değer Fonksiyonu	F(Net)=$\begin{cases} 0 \; if \; net \leq 0 \\ net \; if \; 0 < net < 1 \\ 1 \; if \; net \geq 1 \end{cases}$	Girdiler 0'dan küçük veya eşit ise çıktı 0, 1'den büyük veya eşit ise 1, 0 ve 1 aralığında olduğunda kendi değerini alır.
Sinüs Aktivasyon Fonksiyonu	F(Net)=Sin(Net)	Bu fonksiyon, öğrenme kullanılması planlanan olaylar için sinüs fonksiyonuna uyan bir dağıma sahip olduğunda kullanılmaktadır.

Çıktı: Fonksiyon sonucunda belirlenmiş olan değerdir. Bu çıktı başka bir hücreye gönderilebilir, aynı hücreye girdi olarak da gönderilebilir. Bir elemanın sadece 1 çıktısı olabilir (Pomorova ve Hovorushchenko 2011).

3.6.3 Doğrusal ilkeleme (Linear regression)

Bu analizin amacı bağımlı veya bağımsız değişkenlerin arasındaki ilişkiyi fonksiyon olarak elde etmektir. Ana amacı sebepler ve sonuçlar arasında doğrusal bir ilişki olup olmadığını tespit etmektir (Mihăescu 2011).

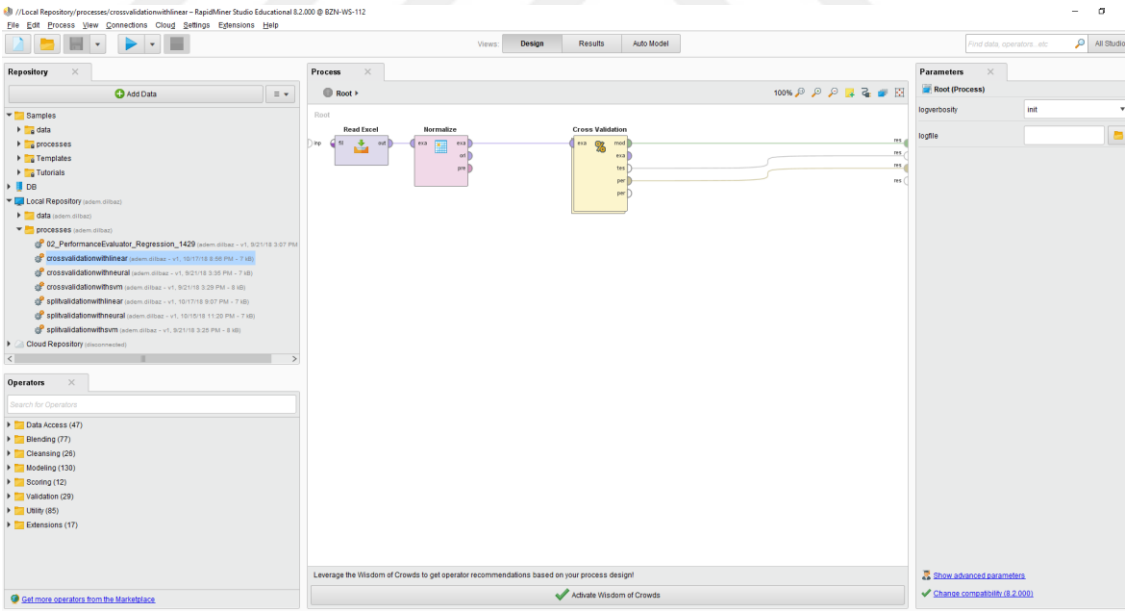


Şekil 3.8 Doğrusal ilkeleme

3.6.4 RapidMiner

RapidMiner, tahminleme, sınıflandırma ve analiz gibi amaçlara yönelik veri madenciliği yapılabilen bir makine öğrenmesi uygulamasıdır. Yazılım ayrıca veri hazırlığı (normalizasyon), makine öğrenmesi uygulama sonuçlarının görselleştirilmesi, validasyon ve optimizasyon gibi adımları da sağlamaktadır (Dwivedi et al 2016).

Bu çalışmada RapidMiner Studio v8.2 yazılımı kullanılmıştır.

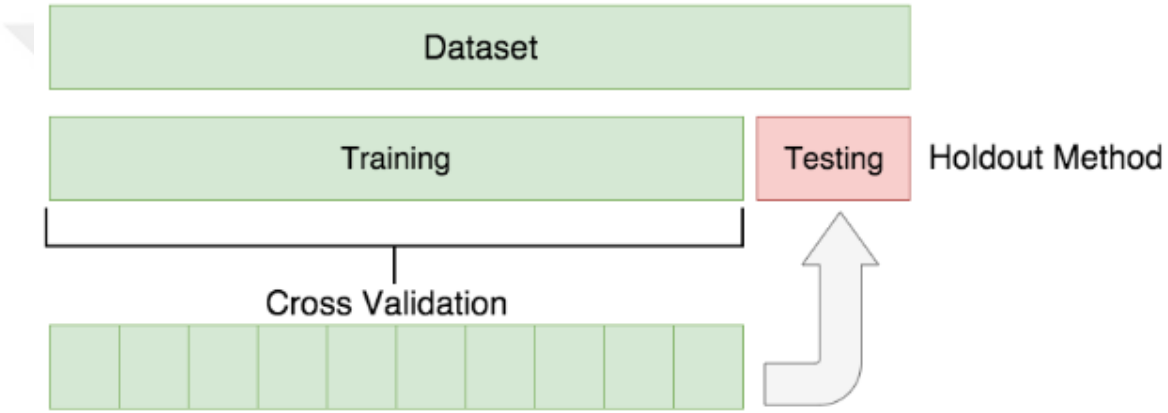


Şekil 3.9 Rapidminer süreç oluşturma arayüzü

3.6.5 Evaluation metrics

Cross Validation (Çapraz Doğrulama)

Veri madenciliğinde, yapılan tahminleme çalışmasının başarı oranının tespiti için veri seti eğitim ve test kümeleri olarak ayrılmaktadır. Çapraz doğrulama, bu ayırma yöntemlerinden biridir. Bu tez çalışmasında çapraz doğrulama yapılan adımlarda veri seti 10 (number of folds) eşit parçaya bölünmüştür (Shanthini and Chandrasekaran 2014).



Şekil 3.10 Çapraz doğrulama şeması

Veri seti 10'a bölündükten sonra 1 parça test kümesi, 9 parça eğitim kümesi olarak ayrılmaktadır. Bu aşamada hangi parçanın test için ayrıldığı bir önem taşımamaktadır, çünkü tüm parçalar sırayla veri seti olarak belirlenip ilgili algoritma çalıştırılarak birer sonuç elde edilmektedir (Wong and Yang 2017). Burada sistemin genel başarı değeri, elde edilen 10 sonucun ortalaması olarak hesaplanmaktadır. Sonuç formülü aşağıdaki gibidir:

$$\frac{\sum_{i=0}^{k} SF(t_i, VK - t_i)}{k}$$

SF: Sınıflandırma fonksiyonu (test veya eğitim)

VK: Veri kümesi

k: katlama sayısı (number of folds), bu çalışmada 10 olarak belirlenmiştir.

t: test kümesi (Wang et al 2007)

Split Validation (Bölünmüş Doğrulama)

Çapraz doğrulamada yaptığımız veri setini k (number of folds) kadar parçaya ayırma işlemi belirtilen oranda ikiye ayırarak yapar. Bölünme oranı 70% ise, veri kümesinin 70% kısmı eğitim, 30% kısmı test amaçlı kullanılır.

3.6.6 Kök ortalama kare hata (RMSE)

Kök ortalama kare hata değerini bu çalışma için yorumlarsak, bir makine öğrenmesi modelinde tahmin edilen değerler ile asıl değerler arasındaki farkın bulunmasında yoğunlukla kullanılan, hata büyüklüğünü ölçen bir metriktir. RMSE tahminleme sonuçlarındaki hatanın standart sapmasıdır, verilere en uygun çekilen çizginin etrafında ilgili veri setinin ne kadar yoğun olduğuna ait değeri verir. 0'dan ∞'a kadar değer alabilir, sıfıra yakın değerler tahminlemenin daha iyi performans gösterdiği anlamına gelir. Sıfır olması o çalışmaya ait modelin hiç hata yapmadığı anlamını taşır. Aşağıdaki formülle hesaplanır (Lakra and Singh 2014):

$$RMSE = \sqrt{\frac{1}{n}\sum_{1}^{n}(\tilde{y} - y)^2}$$

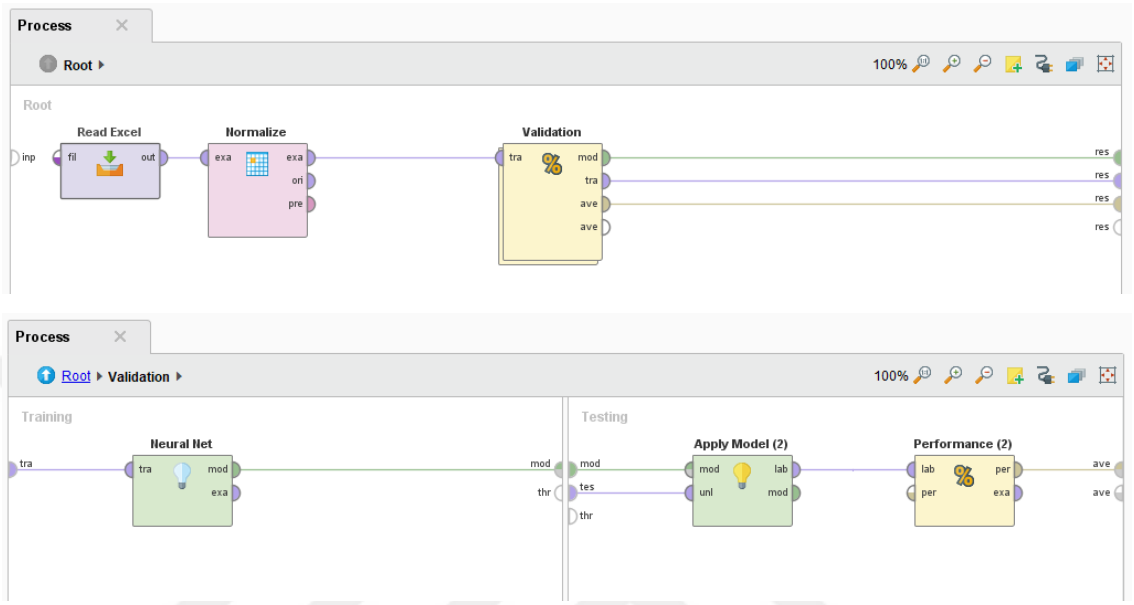
y: gerçek değer

$\tilde{y}$: tahmin edilen değer

3.7 Tez Kapsamında Yapılan Çalışmanın Açıklaması

Bu çalışmada veri seti olarak 102 farklı Java Projesine ait hesaplanmış 13 farklı yazılım metriği kullanılmıştır. Bu sonuçlar bir Excel (Ek 2) dosyasında toplanmış ve RapidMiner uygulamasına girdi olarak verilmiştir. RapidMiner uygulamasında 1 data kaynağı (metrik sonuçları veri seti), 6 süreç (process) oluşturularak tahminleme sonuçları hesaplanmıştır. Veri seti dosyadan okunduktan sonra her süreç için sonuçlar 1'e normalize edilmiş ve sonra validasyon işlemine geçilmiştir. Madde 3.6.5'te açıklanan çapraz doğrulama ve madde 3.6.6.'da açıklanan bölünmüş doğrulama

işlemleri uygulanmıştır. Bu işlemlerde, çapraz doğrulamada "number of folds" değeri 10 olarak alınmış, bölünmüş doğrulamada ise "split ratio" değeri 70% olarak belirlenmiştir.



Şekil 3.11 YSA algoritmasının Rapidminer üzerinde uygulanarak elde edilen sürecin elemanlarının gösterimi

Split validation uygulanarak Neural Network uygulaması süreci

Her iki doğrulama işleminden sonra madde 3.6'da yer alan makine öğrenme uygulamaları Yapay Sinir Ağları, Destek Vektör Makineleri ve Doğrusal İlkeleme sırasıyla uygulanmış, toplamda 6 süreç RapidMiner uygulamasında oluşturulmuştur.
Süreçler sırasıyla çalıştırıldığında bize rms (root mean squared error) sonucunu üretmektedir. Bu çalışmadaki amaç, veri setinin ilgili uygulamalarda çalıştırılması sonucu hata oranını en aza indirmektir. Süreçlerdeki makine öğrenme algoritmalarındaki parametre değerlerindeki artış ve azalışa göre rms değerinin nasıl değiştiği gözlemlenmiş, buna göre hangi sürecin hangi değerlerle çalıştırılması sonucu en az rms değerini verdiği izlenmeye çalışılmıştır.

3.7.1 SVM uygulamasında kullanılan parametreler

- Kernel Type: Bu parametre ile çekirdek fonksiyon türü seçilmektedir.SVM algoritmasında bu çalışmada 4 farklı çekirdek fonksiyon kullanılmıştır.
    - Dot kernel: k(x,y)=x*y
    - Radial kernel: exp(-g ||x-y||^2), g: gama
    - Polynomial kernel: k(x,y)=(x*y+1)^d, d: polynomial degree
    - Multiquadric kernel: $\sqrt{||x-y||^2 + c^2}$

- Karmaşıklık sabiti C: Hatalı sınıflandırma toleransını belirleyen SVM karmaşıklık sabiti
- Converge Epsilon: KKT koşullarında kesinliği belirten optimizasyon parametresi
- L pos: Pozitif veri ve SVM karmaşıklık sabiti için kullanılan bir parametre. Kayıp fonksiyonunun bir parçasıdır.
- L neg: Negatif veri ve SVM karmaşıklık sabiti için kullanılan bir parametre. Kayıp fonksiyonunun bir parçasıdır.
- Epsilon: Duyarsızlık sabitini belirtir. Tahmin değeri gerçek değere yakın olursa bir kayıp olmaz. Kayıp fonksiyonunun bir parçasıdır.
- Epsilon plus: Pozitif sapma için epsilon değerini belirlir. Kayıp fonksiyonunun bir parçasıdır.
- Epsilon minus: Negatif sapma için epsilon değerini belirlir. Kayıp fonksiyonunun bir parçasıdır (Anonymous 2019a).

3.7.2 ANN uygulamasında kullanılan parametreler

Hidden layers: Bu parametre gizli katmanların boyutunu belirtir. Kullanıcı, sinir ağının yapısını bu parametre ile tanımlayabilir. Gerçek düğüm sayısı gizli katmandan daha fazla olmalıdır. Hidden layer değeri -1 olarak verilirse katman boyutu şu şekilde hesaplanır:
(nitelik sayısı + sınıf sayısı) / 2 + 1

Training cycles: Eğitim döngülerinin sayısını belirler. Geri yayılımda, doğru cevap ile çıktı değeri zaten tanımlı olan hata fonksiyonlarının değerlerini hesaplama amaçlı kıyaslanır. Hata daha sonra ağ üzerinden geri beslenmektedir. Bu bilgi doğrultusunda, hata fonksiyonu değerini düşürmek amacıyla her bağlantının ağırlığı düzenlenir. Bu işlem n defa tekrarlanır. Bu parametre kullanılarak n belirtilebilir.

Learning rate: Bu parametre, her adımda ağırlıkları ne kadar değiştirdiğimizi belirler. Learning rate değeri sıfır harici bir değer alabilir.

Momentum: Momentum, önceki ağırlık güncellemesinin bir kısmını şimdiki olana ekler. Bu, yerel maksimumlamayı önler ve optimizasyon yönlerini düzeltir.

Error Epsilon: Eğitim hatası bu epsilon değerinin altına düşerse optimizasyon durdurulur (Anonymous 2019b).

3.7.3 Linear Regression uygulamasında kullanılan parametreler

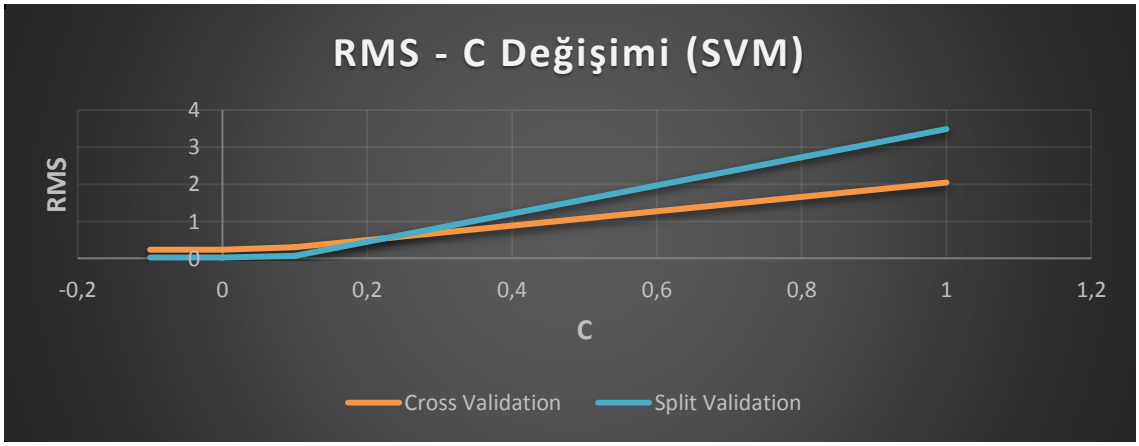
Minimum tolerans: Ortak özellikleri ortadan kaldırmak için minimum toleransı belirler. RapidMiner aracında makine öğrenmesi uygulamalarının çalıştırılması sonrasında uygulama parametreleri ile rms hata değeri arasındaki ilişki bulgular bölümünde yer almaktadır (Anonymous 2019c).

4. BULGULAR

Bu uygulama Intel Core i7-4720HQ, 2.60 GHz CPU ve 16 GB RAM özelliklerine sahip MSI marka bir dizüstü bilgisayarda geliştirilmiştir. Bu bölümde 3 farklı makine öğrenme algoritması kullanılarak 103 adet farklı major versiyonları bulunan Java programlama dilinde yazılmış kaynak kodları bulunan projeler için versiyon tahminlemedeki hata payı sonuçları incelenecektir.

Hesaplanan tüm makine öğrenme algoritması öncesi yazılım metrik sonuçlarının bulunduğu veri seti Excel dosyasından okunarak 1'e normalize edilmiş ve sonrasında 2 farklı tespit (Evaluation) algoritmasına girdi olarak verilmiştir. Bunlar çapraz doğrulama (Cross Validation) ve bölünmüş doğrulamadır (Split Validation). Çapraz doğrulamada veri seti 10 eşit parçaya bölünmüş, bu parçalardan 1 tanesi test seti, 9 tanesi veri seti olarak ayrılmıştır. Bölünmüş doğrulamada ise kullanılan veri setin 70% eğitim verisi, 30% test verisi şeklinde kullanmak amacıyla bölünmüştür. Bu işlemler yapıldıktan sonra Destek Vektör Makineleri, Yapay Sinir Ağları ve Doğrusal İlkeleme algoritmalarına veri setleri verilmiş, sonrasında bu algoritmalara ait parametrelerin değişimleri ile hata oranındaki değişimler gözlemlenmiştir.

Destek Vektör Makineleri (SVM) algoritmasına yazılım metrik sonuçları bulunan veri seti verilmiş ve RMS değerinin SVM parametrelerine göre değişimi gözlemlenmiştir. Buna göre elde edilen sonuçlar aşağıda gösterilmiştir.

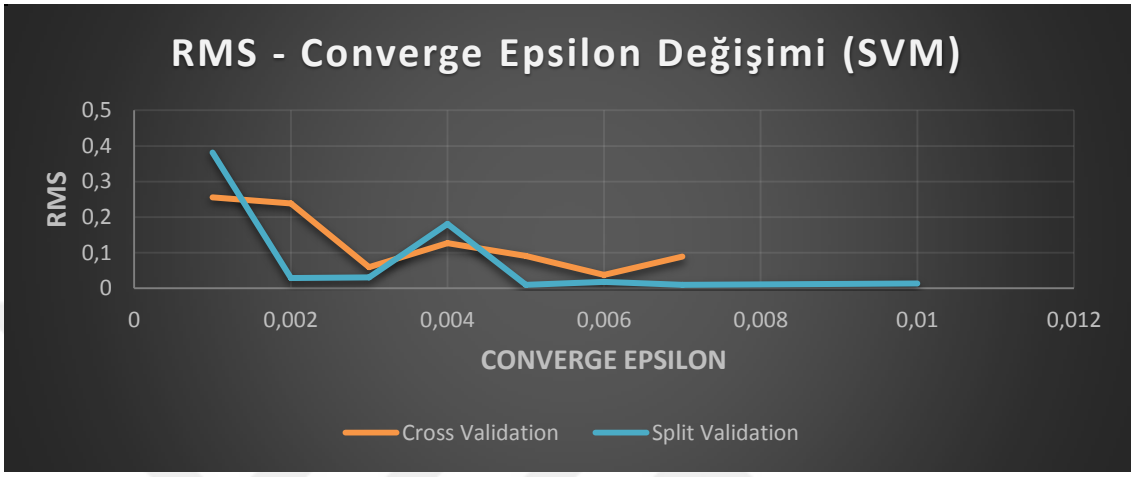


Şekil 4.1 SVM algoritmasında C değerine göre RMS değişimi

Şekil 4.1'e göre Destek Vektör Makineleri için C parametresinin değişiminde minimum RMS değerleri aşağıdaki gibi hesaplanmıştır:

C=-0.1, RMS=0.239 (Cross Validation)

C=-0.1, RMS=0.029 (Split Validation)

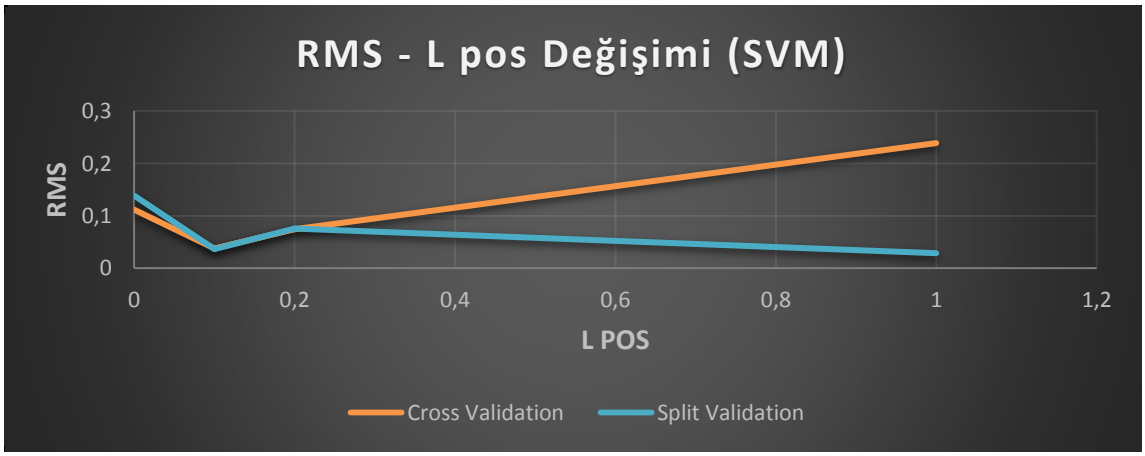


Şekil 4.2 SVM algoritmasında Converge Epsilon değerine göre RMS değişimi

Şekil 4.2'ye göre Destek Vektör Makineleri için Converge Epsilon parametresinin değişiminde minimum RMS değerleri aşağıdaki gibi hesaplanmıştır:

Converge Epsilon=0.006, RMS=0.018 (Cross Validation)

Converge Epsilon=0.005, RMS=0.01 (Split Validation)

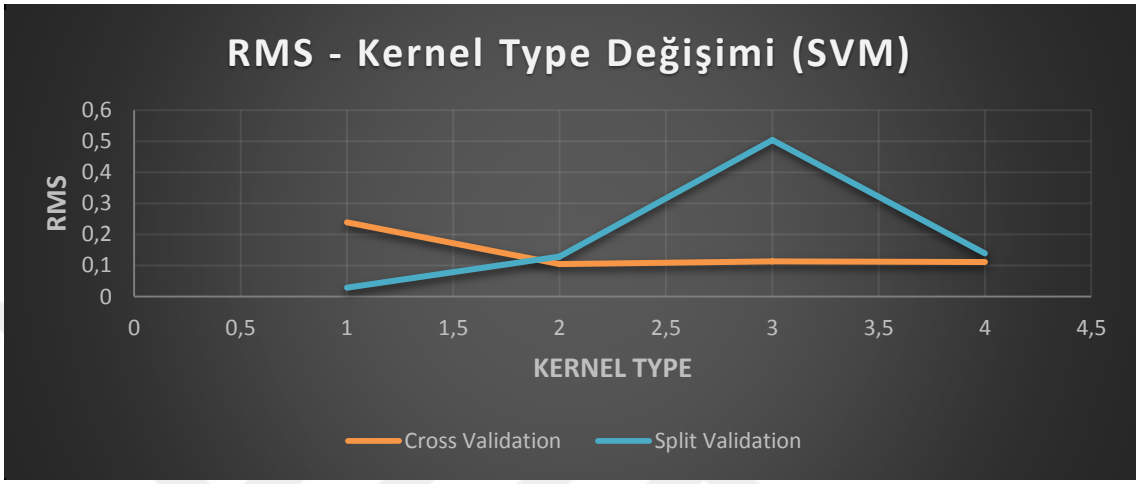


Şekil 4.3 SVM algoritmasında L pos değerine göre RMS değişimi

Şekil 4.3'e göre Destek Vektör Makineleri için L pos parametresinin değişiminde minimum RMS değerleri aşağıdaki gibi hesaplanmıştır:

L pos=0.1, RMS=0.038 (Cross Validation)

L pos=1, RMS=0.029 (Split Validation)

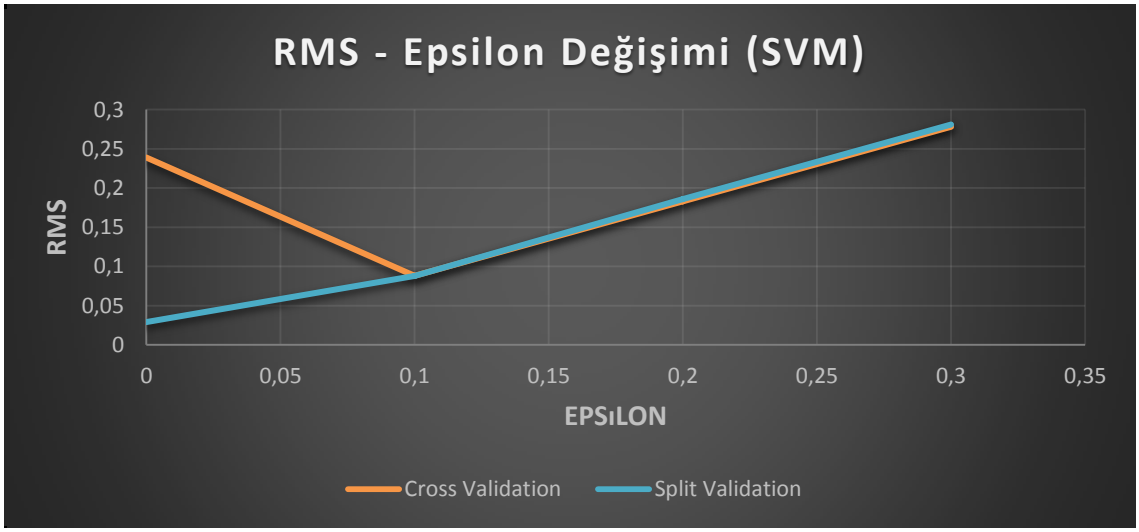


Şekil 4.4 SVM algoritmasında Kernel Type değerine göre RMS değişimi

Şekil 4.4'e göre Destek Vektör Makineleri için Kernel Type parametresinin değişiminde minimum RMS değerleri aşağıdaki gibi hesaplanmıştır:

Kernel Type=multiquadric, RMS=0.112 (Cross Validation)

Kernel Type=dot, RMS=0.029 (Split Validation)

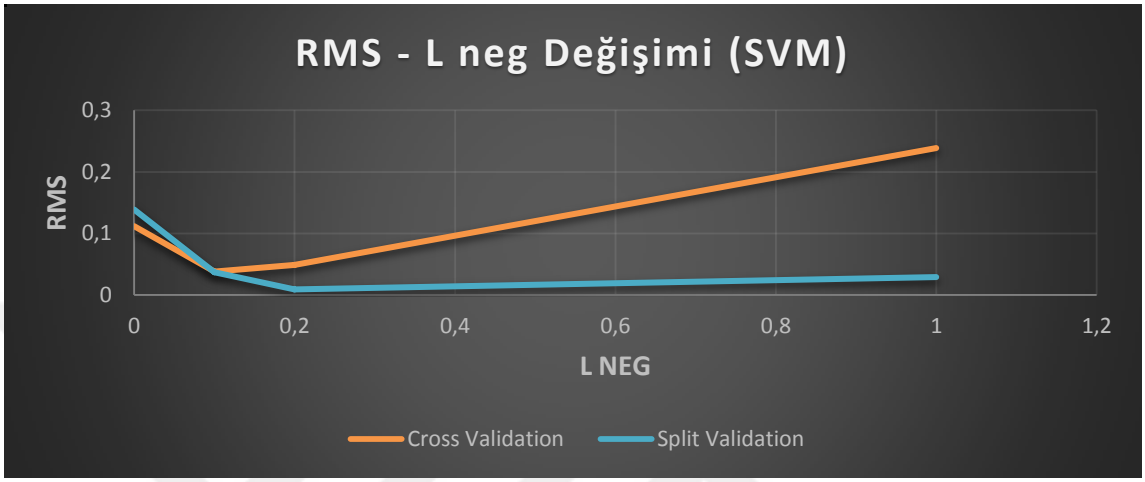


Şekil 4.5 SVM algoritmasında Epsilon değerine göre RMS değişimi

Şekil 4.5'e göre Destek Vektör Makineleri için Epsilon parametresinin değişiminde minimum RMS değerleri aşağıdaki gibi hesaplanmıştır:

Epsilon=0.1, RMS=0.088 (Cross Validation)

Epsilon=0, RMS=0.029 (Split Validation)



Şekil 4.6 SVM algoritmasında L neg değerine göre RMS değişimi

Şekil 4.6'ya göre Destek Vektör Makineleri için L neg parametresinin değişiminde minimum RMS değerleri aşağıdaki gibi hesaplanmıştır:

L neg=0.1, RMS=0.038 (Cross Validation)

L neg=0.2, RMS=0.009 (Split Validation)

Bu sonuçlar doğrultusunda SVM algoritmasında en düşük RMS değerine aşağıdaki parametreler ile ulaşılmıştır.

Doğrulama türü: Bölünmüş Doğrulama

Kernel türü: dot

C: 0

Converge Epsilon: 0.002

L pos: 1

L neg: 0.2

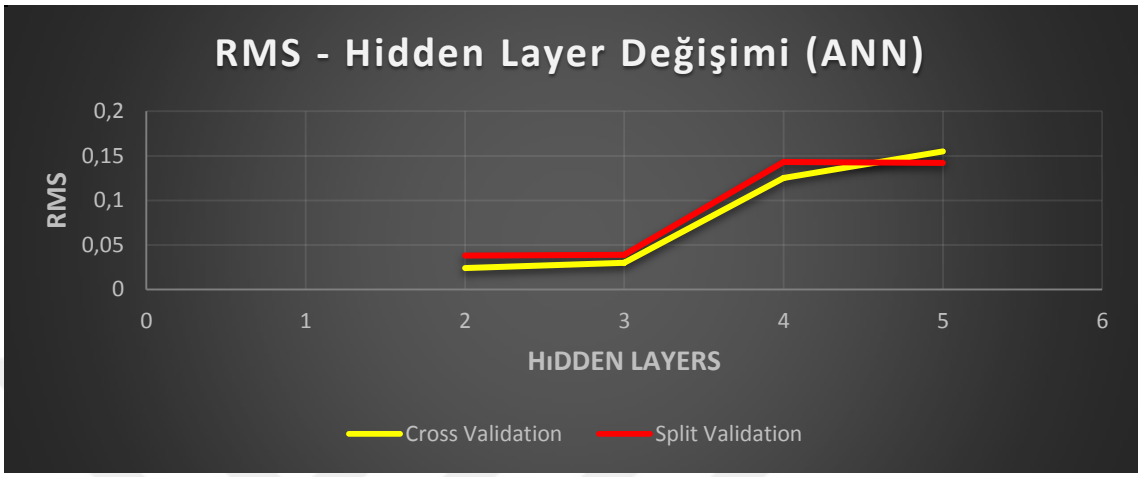
Epsilon: 0

Epsilon plus: 0

Epsilon minus: 0

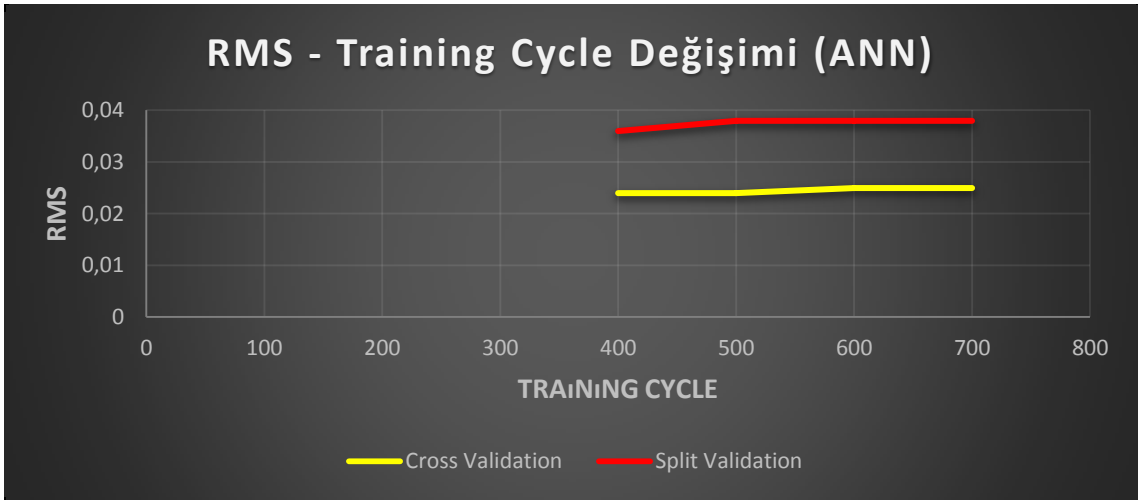
RMS: 0.009

Yapay sinir ağları algoritmasına veri seti verildiğinde 2 farklı doğrulama yöntemi için elde edilen farklı parametrelere göre hata oranları aşağıdaki gibidir.



Şekil 4.7 ANN algoritmasında Hidden layer değerine göre RMS değişimi

Şekil 4.7'ye göre Yapay Sinir Ağları için Hidden Layer parametresinin değişiminde minimum RMS değerleri aşağıdaki gibi hesaplanmıştır:
Hidden Layer=2, RMS=0.024 (Cross Validation)
Hidden Layer=2, RMS=0.038 (Split Validation)

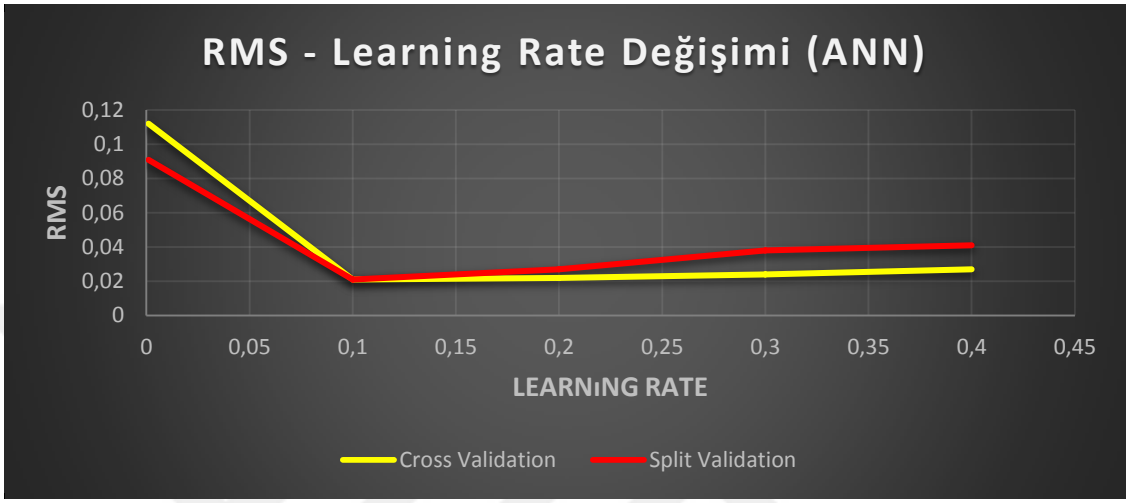


Şekil 4.8 ANN algoritmasında Training Cycle değerine göre RMS değişimi

Şekil 4.8'e göre Yapay Sinir Ağları için Training Cycle parametresinin değişiminde minimum RMS değerleri aşağıdaki gibi hesaplanmıştır:

Training Cycle=400, RMS=0.024 (Cross Validation)

Training Cycle=400, RMS=0.036 (Split Validation)

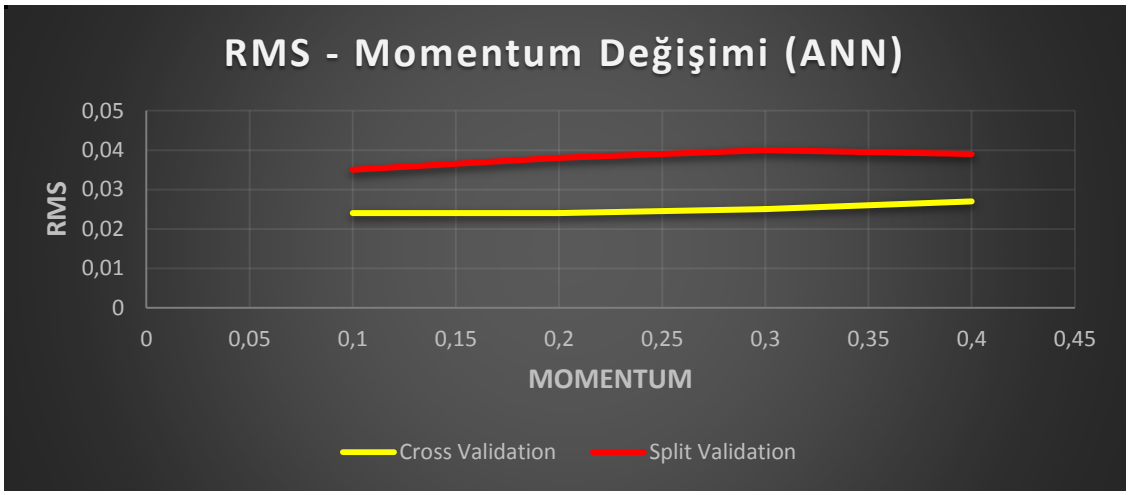


Şekil 4.9 ANN algoritmasında Learning rate değerine göre RMS değişimi

Şekil 4.9'a göre Yapay Sinir Ağları için Learning Rate parametresinin değişiminde minimum RMS değerleri aşağıdaki gibi hesaplanmıştır:

Learning Rate=0.1, RMS=0.021 (Cross Validation)

Learning Rate=0.1, RMS=0.021 (Split Validation)



Şekil 4.10 ANN algoritmasında Momentum değerine göre RMS değişimi

Şekil 4.10'a göre Yapay Sinir Ağları için Momentum parametresinin değişiminde minimum RMS değerleri aşağıdaki gibi hesaplanmıştır:

Momentum=0.1, RMS=0.024 (Cross Validation)

Momentum=0.1, RMS=0.035 (Split Validation)

Bu sonuçlar doğrultusunda yapay sinir ağları algoritmasında en düşük RMS değerine aşağıdaki parametreler ile ulaşılmıştır.

Doğrulama türü: Çapraz Doğrulama

Hidden layer: 2

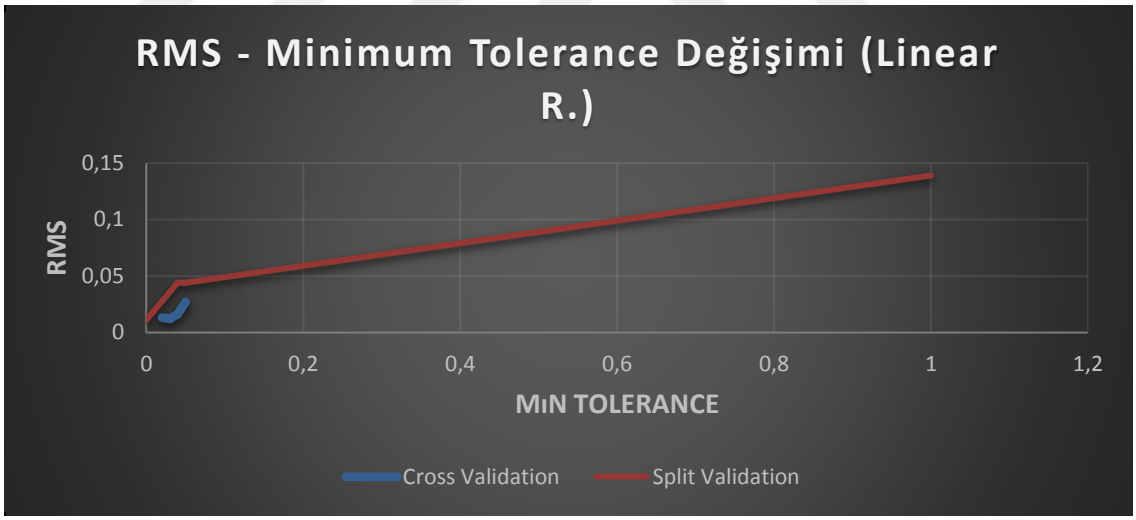
Training cycle: 500

Learning rate: 0.3

Momentum: 0.2

RMS: 0.021

Doğrusal İlkeleme algoritmasına veri seti verildiğinde ise 2 farklı doğrulama yöntemi için aşağıdaki grafikteki sonuçlar elde edilmektedir.



Şekil 4.11 İlkesel Doğrulama algoritmasında Minimum Tolerans değerine göre RMS değişimi

Şekil 4.11'e göre Doğrusal İlkeleme için Minimum Tolerans parametresinin değişiminde minimum RMS değerleri aşağıdaki gibi hesaplanmıştır:

Minimum Tolerans=0.03, RMS=0.012 (Cross Validation)

Minimum Tolerans=0, RMS=0.011 (Split Validation)

Doğrusal ilkeleme algoritmasında sağlanan en küçük hata oranı değerleri ise aşağıdaki gibidir.

Doğrulama türü: Bölünmüş Doğrulama

Minimum tolerans: 0.01

RMS: 0.011

3 farklı makine öğrenme algoritmasının aynı veri seti üzerinde çalıştırılması ile elde edilen sonuçlara göre kullanılan veri seti için en düşük RMS değerinin Destek Vektör Makineleri algoritmasında elde edildiği görülmektedir.

5. SONUÇ ve TARTIŞMA

Bu yüksek lisans tez çalışmasında Java dilinde yazılmış 103 farklı majör versiyonlarına ait kaynak kodu bulunan yazılım projeleri üzerinde gerçekleştirilen yazılım metrikleri hesaplama ve kullanılan yöntemlerin detaylı açıklaması ile makine öğrenme algoritmaları, çalışma prensibi ile ilgili çalışmalardan bahsedilmiştir.

Tez kapsamında, Source Monitor metrik ölçüm uygulaması kullanılarak Maven kütüphanesinden seçilen açık kaynak kodlu Java kütüphaneleri için 13 farklı metrik hesaplaması yapılmıştır. Hesaplanan sonuçlar Rapidminer uygulamasında hazırlanan makine öğrenme algoritma süreçlerine girdi olarak verilmiş ve bu süreçlere dahil edilen tespit (Evaluation) algoritmaları aracılığıyla eğitim ve test veri setleri belirlenen boyutlarda ayrılarak süreç sonucunda RMSE değerlerinin hesaplanması sağlanmıştır. DVM, YSA, DI algoritmalarındaki parametrelerin aldığı değerlerin değişimine göre RMSE değerindeki değişim gözlemlenmiş, hata oranının hangi algoritmada ve hangi parametrelerle minimum değer aldığı tespit edilmeye çalışılmıştır. Buna göre DVM makine öğrenme algoritmasında diğer yöntemlere göre daha az hata payı elde edildiği görülmüştür.

Yazılımda majör versiyon duyurulması, projenin belirli büyüklüklere ulaşması durumunda yapılmaktadır. Bu ölçümlerin yapılmasındaki en etkin yöntem ise yazılım metriklerinin belirli periyotlarda hesaplanmasıdır. Makine öğrenmesi, verilen bilgiyi öğrenerek anlamlı veriyi ortaya çıkarır. Bu çalışmada her bir projenin farklı majör versiyonları için metrikler hesaplanıp, gelecek majör versiyon için yapılan tahminlemenin ne kadar hata payına sahip olabileceği bilgisine farklı makine öğrenmesi algoritmalarıyla ulaşılmaya çalışılmıştır. Makine öğrenmesinin mevcut hesaplama ve veri miktarlarındaki artıştan kolayca yararlanabilmesi ve insan gücünü minimum seviyelere indirmesi nedeniyle günümüzde ve yakın gelecekte çok daha fazla başarıya sahip olacağı ve yazılım dünyasında daha yoğunlukla kullanılacağı düşünülmektedir. Yapılan çalışmanın geliştirilmesi amacıyla hesaplamaların bilinen metrik kümeleri üzerinden metriklerin hesaplanması ya da hesaplamaya minör versiyonların da dahil edilerek kapsamın genişletilmesi sağlanabilir.

KAYNAKLAR

Anandhi, V., Chezian R. 2014. Regression Techniques In Software Effeort Estimation Using COCOMO Dataset. International Conference on Intelligent Computing Applications, pp.353-357.

Anonymous. 2017a. Web Sitesi: http://ecomputernotes.com/software-engineering /classification-of-software-metrics, Erişim Tarihi: 6.01.2017.

Anonymous. 2017b. Web Sitesi: http://serhatdirik.blogspot.com.tr/2011/04/yazlm-olcumleme.html, Erişim Tarihi: 10.04.2017.

Anonymous. 2017c. Web Sitesi: http://nightwalkers.blogspot.com.tr/2010/05/kod-kalite olcumleri-ve-sk-kullanlan.html, Erişim Tarihi: 10.06.2017.

Anonymous. 2017d. Web Sitesi: http://www.cihataltuntas.com/ codemetricscyclomatic complexity, Erişim Tarihi: 10.03.2017.

Anonymous. 2017e. Web Sitesi: http://web.itu.edu.tr/buzluca/ykgs08_2.pdf, Erişim Tarihi: 19.05.2017.

Anonymous. 2017f. Web Sitesi: http://support.objecteering.com/objecteering6.1/help/ us/metrics/metrics_in/specialization_index.htm, Erişim Tarihi: 05.01.2017.

Anonymous. 2017g. Web Sitesi: http://findbugs.sourceforge.net/index.html, Erişim Tarihi: 05.01.2017.

Anonymous. 2018a. Web Sitesi: http://yazilimagiris.com/2017/12/yazilim-urun-metrikleri/, Erişim Tarihi: 05.07.2018.

Anonymous. 2018b. Web Sitesi: web.firat.edu.tr/mbaykara/8.Hafta-GözdenGecirme Teknikleri.pdf, Erişim Tarihi: 05.07.2018.

Anonymous. 2018c. Web Sitesi: bm.erciyes.edu.tr/mcelik/bz313/Bolum05_Yazilim_ Surec_Proje_Metrikleri.ppt, Erişim Tarihi: 05.07.2018.

Anonymous. 2018d. Web Sitesi: https://slideplayer.biz.tr/slide/2982443/, Erişim Tarihi: 05.07.2018.

Anonymous. 2019a. Web Sitesi: https://docs.rapidminer.com/latest/studio/operators/ modeling/predictive/support_vector_machines/support_vector_machine.html, Erişim Tarihi: 25.03.2019.

Anonymous. 2019b. Web Sitesi: https://docs.rapidminer.com/latest/studio/operators/ modeling/predictive/neural_nets/neural_net.html, Erişim Tarihi: 25.03.2019.

Anonymous. 2019c. Web Sitesi: https://docs.rapidminer.com/latest/studio/operators/ modeling/predictive/functions/linear_regression.html, Erişim Tarihi: 25.03.2019.

Beratoğlu, M., Buzluca, F.2009. Görselleştirme Tekniklerinin Yazılım Metriklerine Uygulanması, 4. Ulusal Yazilim Muhendisligi Sempozyumu, 173-197, İstanbul, Türkiye.

Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R. and Selby, R. 1995. Cost models for future software life cycle processes: COCOMO 2.0. Annals of Software Engineering, 1(1), pp.57-94.

Bunge, M. (1981). Treatise on Basic Philosophy. Ontology I :The Furniture of the World, 41(4), p.565.

Can, H., Jianchun X., Ruide L., Qiliang Y and Liqiang X. 2013. A New Model for Software Defect Prediction Using Particle Swarm Optimization and Support Vector Machine. 25th Chinese Control and Decision Conference (CCDC), pp.4106-4110.

Catal, C., Alan, O. and Balkan, K. 2011. Class noise detection based on software metrics and ROC curves. Information Sciences, 181(21), pp.4867-4877.

Dwivedi, S., Kasliwal P. and Soni S. 2016. Comprehensive Study of Data Analytics Tools (RapidMiner, Weka, R tool, Knime). 2016 Symposium on Colossal Data Analysis and Networking (CDAN).

Eisty, N., Thiruvathukal G. and Carver J. 2018. A Survey of Software Metric Use in Research Software Development. 2018 IEEE 14th International Conference on e-Scienc, pp.212-222.

Hill, P. 2011. Practical software project estimation. New York: McGraw-Hill.

Kitchenham, B. 2010. What's up with software metrics? – A preliminary mapping study. Journal of Systems and Software, 83(1), pp.37-51.

Lakra, S. and Singh, N. 2014. An activeness metric for the quality of design of a Multilayer Perceptron Model. 1-5. 10.1109/ICDMIC.2014.6954223.

Lanza, M. and Marinescu, R. 2011. Object-oriented metrics in practice. Berlin: Springer.

Lincke, R., Lundberg, J. and Löwe, W. 2008. Comparing software metrics tools. Proceedings of the 2008 international symposium on Software testing and analysis - ISSTA '08.

Mihăescu, M. 2011. Classification of Learners Using Linear Regression. Proceedings of the Federated Conference on Computer Science and Information Systems, pp.717-721.

Núñez-Varela, A., Perez-Gonzalez, H., Cuevas-Tello, J. and Soubervielle-Montalvo, C. 2013. A Methodology for Obtaining Universal Software Code Metrics. Procedia Technology, 7, pp.336-343.

Palıgu, F., Yağcı, N. and Öztürk S. 2013. Kodu İyileştirmeye Nereden Başlamalı? Bir Yazılım Metrik Yaklaşımı: Yazılım Kalite Risk Oranı. Proceedings of the 7th National Software Engineering Symposium, İzmir.

Pomorova, O., Hovorushchenko, T. 2011. Research of Artificial Neural Network's Component of Software Quality Evaluation and Prediction Method. The 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications 15-17 September 2011, Prague, Czech Republic, pp.959-962.

Radjenović, D., Heričko, M., Torkar, R. and Živkovič, A. 2013. Software fault prediction metrics: A systematic literature review. Information and Software Technology, 55(8), pp.1397-1418.

Rajper, S. and Shaikh, Z. 2016. Software Development Cost Estimation: A Survey. Indian Journal of Science and Technology, 9(31)

Shanthini, A., Chandrasekaran R. 2014. Analyzing the Effect of Bagged Ensemble Approach for Software Fault Prediction in Class Level and Package Level Metrics. ICICES2014 - S.A.Engineering College, Chennai, Tamil Nadu, India.

Subramanian, G. and Corbin, W. 2001. An empirical study of certain object-oriented software metrics. Journal of Systems and Software, 59(1), pp.57-63.

Tanriover, O. and Eryigit, R. 2015. An emprical analysis of early object oriented design metrics in relation to code size. 2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS).

Voulgaropoulou, S., Spanos G. and Angelis L. 2012. Analyzing Measurements of the R Statistical Open Source Software. IEEE 35th Software Engineering Workshop.

Wang, L., HU, X., Ning, Z. and Ke, W. 2007. Predicting Object-Oriented Software Maintainability using Projection Pursuit Regression. The 1st International Conference on Information Science and Engineering (ICISE2009), pp.3827-3830.

Wong, T. and Yang, N. 2017. Dependency Analysis of Accuracy Estimates in k-Fold Cross Validation. IEEE Transactions on Knowledge and Data Engineering, 29(11), pp.2417-2427.

EKLER

EK 1 Kullanılan Kütüphaneler

EK 2 Metrik Sonuçları

EK 1 Kullanılan Kütüphaneler

Finagle Kütüphaneleri

finagle-core	finagle-http	finagle-memcached	finagle-serversets
finagle-stats	finagle-thrift	finagle-zipkin	

Jetty Kütüphaneleri

jetty-example-embedded	jetty-annotations	jetty-client	jetty-deploy
jetty-http	jetty-io	jetty-jaspi	jetty-jmx
jetty-jndi	jetty-plus	jetty-rewrite	jetty-security
jetty-server	jetty-servlet	jetty-start	jetty-webapp
jetty-util	jetty-xml	jetty-test-webapp	

Spring Kütüphaneleri

spring-aop	spring-aspects	spring-beans	spring-context
spring-context-support	spring-core	spring-expression	spring-instrument
spring-instrument-tomcat	spring-jdbc	spring-jms	spring-messaging
spring-orm	spring-oxm	spring-test	spring-tx
spring-web	spring-webmvc	spring-webmvc-portlet	spring-websocket

spring-security-config	spring-security-core	spring-security-taglibs	spring-security-web

Hibernate Kütüphaneleri

hibernate-c3p0	hibernate-commons-annotations	hibernate-core	hibernate-ehcache
hibernate-entitymanager	hibernate-envers	hibernate-infinispan	hibernate-jpamodelgen
hibernate-proxool	hibernate-search-engine	hibernate-search-infinispan	hibernate-search-orm
hibernate-testing	hibernate-tools	hibernate-validator	hibernate-validator-annotation-processor

Maven Kütüphaneleri

maven-archiver	maven-artifact	maven-core	maven-embedder
maven-model	maven-plugin-api	maven-project	maven-settings

Openjpa Kütüphaneleri

openjpa	openjpa-all	openjpa-jdbc	openjpa-kernel
openjpa-lib	openjpa-persistence	openjpa-persistence-jdbc	openjpa-persistence-locking
openjpa-slice	openjpa-xmlstore		

Tomcat Kütüphaneleri

tomcat-api	tomcat-catalina	tomcat-catalina-ant	tomcat-catalina-ha
tomcat-dbcp	tomcat-el-api	tomcat-jasper	tomcat-jdbc
tomcat-jni	tomcat-jsp-api	tomcat-juli	tomcat-servlet-api
tomcat-storeconfig	tomcat-tribes	tomcat-util	tomcat-util-scan
tomcat-websocket	tomcat-websocket-api		

EK 2 Metrik Sonuçları

A: Number of Files

B: Number of Statements

C: % of Branches

D: Number of Calls

E: % of Comments

F: Number of Classes

G: Average Depth

H: Methods / Classes

I: Average Statements / Methods

J: Maximum Complexity

K: Average Complexity

L: Number of Methods

M: Number of Lines

No	A	B	C	D	E	F	G
1	227.0	7450.7	10.5	3061.3	38.1	241.7	1.6
2	8.0	410.8	14.4	166.5	27.8	9.0	1.7
3	311.0	8814.0	11.2	3674.3	42.7	330.8	1.6
4	97.7	6829.7	10.9	3426.7	27.8	139.7	1.7
5	256.5	12620.0	13.8	5586.0	46.0	273.5	1.8
6	403.3	16545.0	12.5	7449.7	43.5	440.0	1.7
7	177.3	5270.0	13.2	2116.0	49.7	183.0	1.8
8	201.0	9170.0	9.4	4129.0	41.4	194.3	1.5
9	109.0	6572.7	15.0	2932.3	41.4	161.3	2.2
10	157.0	6388.5	11.7	2845.0	34.8	172.0	1.7
11	99.7	5219.0	15.9	2228.7	43.1	121.0	2.2
12	242.0	9491.7	13.5	4335.7	46.7	262.3	2.0
13	103.7	5613.0	20.4	3149.7	29.6	121.0	2.3
14	319.3	17793.3	19.4	8100.7	42.1	384.0	2.2
15	78.7	3542.0	12.1	1383.3	42.5	80.7	1.7
16	547.3	18636.0	12.9	7812.3	44.9	558.0	1.7
17	207.3	7644.7	13.4	3164.7	42.1	245.3	1.8
18	286.3	14894.7	17.0	7072.3	41.9	337.7	2.2
19	176.5	17906.5	29.75	8569	33.05	235	2.765
20	72	8926.5	16.45	3162	52.5	185.5	2.625
21	14	3188.5	29.2	1298	8.2	24	2.465

22	46	1462.5	12.5	754	47.85	47	1.51
23	78	4206.5	15.4	1777.5	30.1	102	2.27
24	7	497.5	13.3	256.5	23.6	25	2.98
25	36.5	2874.5	15.2	1376.5	17.85	57.5	2.46
26	7	778	17.9	312.5	25	14	2.16
27	41.5	2146	9.45	1050	29.75	82	2.155
28	31.5	528.5	17.3	188	3.8	14	1.625
29	18	354.5	9.65	133	38.25	20	1.575
30	14.5	1096.5	23.15	634	19.45	29	2.455
31	24.5	2229.5	18.8	986.5	18.4	37	2.47
32	14.5	846	13.9	401	30.35	19.5	1.85
33	29	4357.5	25.05	1786.5	27.05	40.5	3.065
34	31.5	3151	19.7	1338	24.15	44	2.62
35	4.5	609.5	21.65	382	16.8	5	2.935
36	9.5	1021.5	19.25	642.5	28.65	15.5	2.56
37	38	2245	15.35	1112.5	26.8	42.5	2.05
38	17	472	9.7	133	46.65	17.5	1.555
39	33.5	2021	15.6	774.5	25.05	53	2.145
40	64.5	10111	18.9	4504.5	25.9	121	2.555
41	15	2568.5	21.9	1425	20.2	27.5	2.5
42	5.5	936	20.95	692	14.85	8.5	3.405
43	60.5	8036.5	24.6	3436	21.05	102.5	2.765
44	18.5	2435	24.7	1505	22.1	26.5	2.745
45	3	950.5	25.7	520	19.65	8.5	3.13
46	12.5	1322	15	994.5	12.15	16.5	2.56
47	79	6695	12.85	3699	31.35	105	2.2
48	3.5	1217	5.85	328.5	13	3	1.53
49	46	1569.5	10.5	710	34.95	51	1.415
50	2456.5	144422	14.75	67851	30.65	2972.5	1.985
51	50	3928	9.85	1485	27.3	52	1.675
52	204.5	12113	12.4	5545.5	22.1	311	1.795
53	227.5	9465	9.2	3970.5	26.5	237.5	1.585
54	37	1979.5	13.55	862	27.05	42	1.995
55	123	5614.5	8.25	1400.5	54.9	122.5	1.465
56	3.5	1228.5	5.3	317.5	17.65	3	1.51
57	494	20022	11.9	7703.5	30.25	535	1.71
58	7	1382.5	0.75	593	5.95	6.5	1.49
59	57.5	3335	11.1	1276.5	27.05	67	1.62
60	64	2210.5	10.35	890.5	26.7	93	1.625
61	138.5	11353	17.25	6679.5	14.7	167.5	2.17
62	193.5	9357	13.35	3928.5	31.6	213	1.715
63	21.5	896	13.25	440	36.55	34.5	1.615
64	33.5	2467	14.45	1111.5	27.25	50.5	1.955

65	54	3809	17.2	2275.5	36.55	84	2.365
66	4.5	345.5	13.05	193.5	27.6	4.5	2.31
67	44	1665.5	15.85	739.5	23.75	46	2.16
68	172.5	8893	14.9	4585	20.15	192	2.04
69	9	753.5	14	467	16.05	10	1.85
70	47.5	6468	25.4	5852.5	33.05	47.5	2.805
71	16.5	805.5	6.05	440	33.55	16.5	1.62
72	52	4234	16.85	2772.5	18	54.5	2.25
73	18.5	1697	20.5	1378	36.35	18.5	2.535
74	1337.5	152437.5	21.6	78714.5	26.85	1765.5	2.345
75	2099	200958	21.05	98622.5	33.2	2684.5	2.31
76	357.5	47492	19.7	25718.5	25.15	499.5	2.13
77	489.5	63961	24.9	31980	26.2	613	2.65
78	156.5	14571	16.1	6543.5	29.85	208.5	2.1
79	234	18816.5	18.25	10244.5	31.1	371	2.075
80	65	4360.5	27.25	2691.5	25.4	24	2.325
81	30	2454.5	14.75	1072.5	23.8	42.5	1.75
82	5	782	32.7	464	27.85	7	2.92
83	492	32861	13.8	18011	38.2	560	2.055
84	382	52225	20.85	28413.5	30.8	621	2.44
85	30	1447.5	17.9	566.5	43.15	30	2.17
86	39.5	3860.5	18	1974.5	33.3	44.5	2.25
87	77	11267.5	16	4955	37.25	101.5	2.26
88	34	1984	20.3	899.5	24.1	53	2.465
89	117	20360.5	20.15	11499.5	26.4	248	2.795
90	31.5	4548.5	16.3	2261.5	31.45	47.5	2.24
91	36	1678.5	6.9	677	62.15	41.5	1.63
92	46	1315.5	10.65	394.5	64	59	2
93	11.5	949.5	19	496	35.9	23	2.43
94	73.5	1777	7.45	516	68.65	71.5	1.35
95	37.5	1772	19	985	39.85	37.5	2.13
96	87.5	8489.5	16.25	4131	30.9	127.5	2.255
97	60.5	5410.5	21.15	2409	31.6	85	2.465
98	76.5	6574.5	19.75	3284.5	28.45	106.5	2.255
99	61.5	4609	19.55	2324	18.35	92.5	2.355
100	33	575.5	10.75	62.5	44.15	46	1.435
101	31.5	2266.5	12.7	961.5	27.35	62	1.705
102	14	399.5	8.75	156	47.55	15	1.73

No	**H**	**I**	**J**	**K**	**L**	**M**
1	5.0	3.2	33.0	1.9	1173.0	22404.3
2	6.7	3.7	16.5	2.2	60.0	1031.5

3	4.1	3.2	24.5	1.9	1369.5	27800.8
4	5.1	6.6	25.7	2.5	766.7	17608.0
5	7.2	3.6	40.0	2.0	1962.5	41882.0
6	6.3	3.1	89.7	2.0	2791.0	54001.0
7	4.9	3.2	43.3	1.9	903.0	20171.0
8	9.5	2.7	25.3	1.6	1773.7	29905.3
9	7.1	3.6	45.3	2.2	1061.7	21147.3
10	6.2	3.0	22.5	1.8	1096.5	18363.5
11	7.1	3.8	34.0	2.2	859.7	16986.3
12	6.9	2.9	46.3	2.0	1794.7	34193.0
13	5.9	5.3	101.3	3.1	713.0	13725.3
14	6.6	4.4	95.0	2.7	2567.3	53614.7
15	8.0	3.0	17.7	1.8	631.0	11480.0
16	5.2	3.3	98.3	2.1	2900.0	63304.3
17	5.1	3.3	27.3	2.0	1259.3	24796.3
18	6.4	4.2	58.3	2.6	2175.3	46668.3
19	7.425	7.88	219	3.58	1745.535	42814.5
20	9.32	3.91	28.5	2.14	1750.22	28577.5
21	17.37	5.495	42	2.88	418.98	5308.5
22	6.475	2.57	8.5	1.725	307.145	4990
23	6.695	3.72	40	2.205	684.75	9795
24	2.855	4.875	13	2.42	71.56	1065
25	7.895	4.27	49	2.15	450.4	5509
26	11.29	3.04	21	2.635	158.06	1671
27	5.6	2.705	32	1.735	455.57	5411.5
28	4	5.17	6	1.81	56	1085
29	3.26	2.71	5.5	1.655	64.03	1016.5
30	4.545	6.94	23	3.53	118.015	2697.5
31	8.895	4.525	60	2.42	330.01	5127
32	8.39	3.26	35	2.005	153.075	2301
33	11.32	7.205	229.5	4.045	460.485	10877
34	11.19	4.715	115	3.145	490.52	7691
35	9.55	11.515	54	5.435	41.51	1257.5
36	9.805	4.76	61.5	2.645	152.015	3052
37	6.6	5.22	22.5	2.3	280.485	6024
38	5.695	2.225	11	1.59	99.51	1862.5
39	6.485	3.46	29.5	2.33	344.875	5073
40	12.515	4.675	66.5	2.655	1514.105	24886
41	10.42	6.685	78	3.52	284.93	5740.5
42	6.6	14.15	27.5	4.445	57.5	2006
43	10.115	5.915	99	3.695	1034.775	18508
44	10.82	7.025	64	4.13	274.605	5704
45	9.385	9.94	46.5	4.8	77.995	1947

46	5.725	11.22	106	3.555	93.5	2516
47	9.155	4.065	69	1.965	962.9	19274
48	148.17	2.905	11.5	1.89	444.51	2531.5
49	8.17	2.095	8	1.55	415	4492.5
50	8.45	3.565	261	1.945	24749.86	371476
51	19.04	2.33	12	1.485	964.61	10295.5
52	8.225	2.59	32.5	1.735	2498.885	29516
53	7.385	2.845	23	1.635	1746.22	24359
54	7.545	3.63	27	2.005	312.05	4540.5
55	9.475	2.09	19	1.43	1160.685	22405
56	151.67	2.235	9.5	1.8	455.01	2583.5
57	6.555	2.99	25	1.825	3509.2	49425
58	74.665	1.68	4	1.03	416.985	2601
59	8.86	2.93	14	1.775	586.9	7945
60	4.41	2.63	19	1.6	416.93	5427.5
61	9.695	4.77	24.5	2.44	1621.475	22067.5
62	6.09	3.815	22	2.075	1367.98	24267.5
63	3.255	4.23	24.5	2.32	110.555	3057
64	8.15	3.57	55	2.28	417.45	6368.5
65	7.37	3.735	33.5	2.275	619.36	9989.5
66	11.325	4.435	29.5	1.97	52.5	1062.5
67	7.65	2.96	28.5	2.415	336	4835
68	6.785	4.95	49.5	2.565	1377.865	24717.5
69	10.665	5.275	29	2.455	99.98	1892
70	15.545	6.845	80	3.345	747.975	18808
71	9.06	2.215	12	1.365	168.02	2595.5
72	11.205	4.825	28	2.515	603.18	11080
73	11.605	6.105	39.5	2.76	212.96	5171.5
74	13.83	4.41	607	2.755	24426.66	328043
75	12.36	4.22	607	2.59	33212.91	478631
76	14.91	4.515	535.5	2.84	7448.175	102579
77	15.49	5.055	447	3.07	9500.17	129850
78	14.365	3.17	140	2.08	2996.315	34565.5
79	9.405	3.27	113	2.285	3489.78	45045
80	16.125	7.9	152	4.585	387	9119.5
81	12.83	2.66	13	1.745	546.895	5309
82	8.43	10.27	21	3.975	59.01	1575.5
83	8.265	4.64	47	2.11	4669.875	102862
84	9.815	6.27	307	3.26	6097.68	143586
85	7.935	3.91	27	2.29	236.5	4683
86	12.43	4.8	30	2.55	552.985	9615
87	18.365	4.4	49.5	2.135	1863.82	27563.5
88	4.745	5.4	29	2.805	251.595	4756

89	8.455	7.155	104.5	3.42	2096.285	45964
90	18.535	3.395	60	2.155	880.045	11578
91	3.065	3.76	82.5	2.895	132.45	6748
92	4.52	2.555	17	1.655	266.68	7304
93	4.765	6.055	21.5	3.005	109.54	2593.5
94	8.2	1.165	25	1.51	585.775	11154
95	5.32	5.785	17	3.035	199.47	5094
96	10.38	4.38	31	2.41	1323.45	18435.5
97	7.475	6.22	123.5	3.155	652.205	13644.5
98	8.53	5.105	143	2.61	908.41	17119.5
99	5.195	6.2	70	3.055	480.695	10250.5
100	4.305	0.365	5	1.11	198.03	2080
101	8.39	2.345	15	1.765	520.66	5366
102	6.93	2.125	6	1.54	103.95	1415

ÖZGEÇMİŞ

Adı Soyadı : Adem Dilbaz

Doğum Yeri : Çankırı

Doğum Tarihi :14.08.1989

Medeni Hali : Evli

Yabancı Dili : İngilizce

Eğitim Durumu

Lise	Hayrullah Kefoğlu Anadolu Lisesi (2007)
Lisans	Çankaya Üniversitesi Bilgisayar Mühendisliği, Çift Anadal (2013)
	Çankaya Üniversitesi Matematik- Bilgisayar Bölümü, Anadal (2013)
Yüksek Lisans	Ankara Üniversitesi Fen Bilimleri Enstitüsü
	Bilgisayar Mühendisliği Anabilim Dalı (2020)

Çalıştığı Kurumlar

Biznet Bilişim (2013 - Halen)