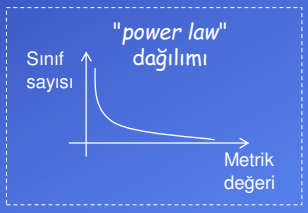
Tasarım kusurlarının makine öğrenmesi tabanlı yöntemlerle belirlenmesi

Metrikler ile kural tabanlı çalışmanın zorlukları:



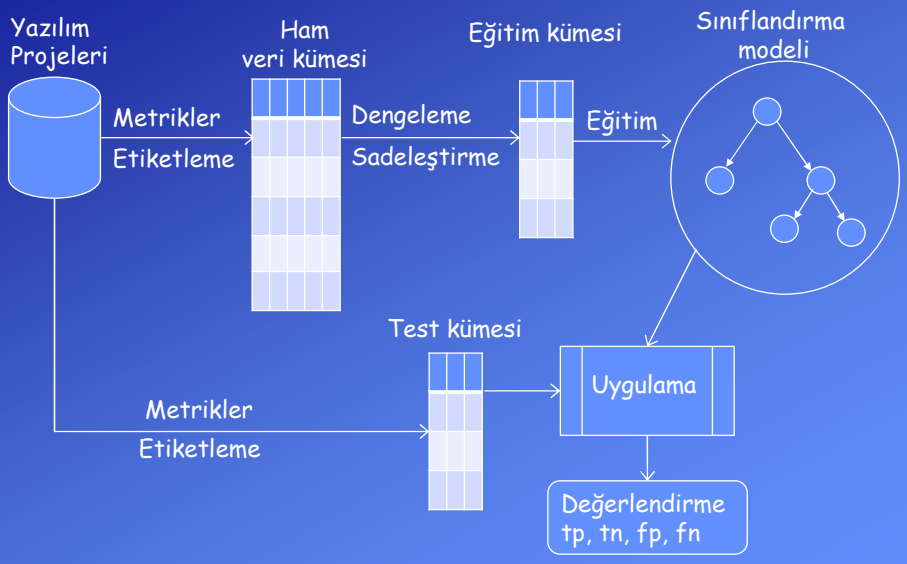
- Metriklerin değerleri genelde "*power law*" dağılımı gösterir. Çok sayıda sınıf aynı değere sahip olabilir.

  Bu dağılımda ortalama, standart sapma gibi istatistiksel yöntemlerle normal değerler ve eşik değerleri belirlemek sağlıklı sonuçlar vermez.
- Metriklerin belli bir kusurla ilgili olarak eşik değerlerini belirlemek zordur.
- Metriklerin değer aralıkları çok farklıdır [1 , ∞], [0 , 1]. Bunları aynı denklem içinde kullanmak güçtür. Çeşitli ölçekleme, normalizasyon işlemleri gerekir.
- Belli bir tasarım kusuru ile hangi metriğin hangi oranda ilişkili olduğunu (kuralları) belirlemek zordur.

Makine öğrenmesi tabanlı yöntemler:

- Kural tabanlı yöntemlerde, uygun metriklere karar vermek, referans değerleri belirlemek ve kuralları oluşturmak yöntemi tasarlayan kişiler tarafından bilgi ve deneyime bağlı olarak yapılır.
- İnsan eforunu azaltmak için makine öğrenmesine dayalı yöntemler oluşturulur.
- Gerçek projelerden kusurlu (ve kusursuz) sınıflar ile ilgili bilgiler (metrikler) toplanıp öğrenebilen bir model bu veriler ile eğitilebilir.
- Daha sonra bu model kusurlu sınıfların belirlenmesi için kullanılabilir.
- Bu yöntemlerde güvenilir, sağlıklı veri kümelerine gerek vardır.

Öğrenme tabanlı yöntemlerin oluşturulması



Öğrenme tabanlı yöntemlerin oluşturulması

1. Amacın belirlenmesi

Aşağıdaki gibi farklı amaçlara yönelik yöntemler oluşturulabilir.

- Belli bir kusurun bulunması (*defect detection, smell detection*)

  Tanımlı bir kusur (God Class, Data Class gibi) kurulacak olan sisteme öğretilip bulunmaya çalışılır.
- Hata çıkarma olasılığı yüksek birimlerin bulunması (*error-prone modules*)

  Yapısal kusurları olduğundan testlerde veya kullanım sırasında hata çıkarmaya meyilli olan sınıflar (metotlar), hata çıkmadan önce belirlenmeye (öngörülmeye) (*prediction*) çalışılır.
- Değişme olasılığı yüksek olan birimlerin bulunması (*change-prone modules*)

  Yazılım birimleri (sınıflar, metotlar) iki ana nedenden dolayı değişirler.

  a) Hataların düzeltilmesi (*bug fix*)

  b) Kullanıcıdan gelen değişiklik istekleri (*change request*)

  Değişimler projenin maliyetini arttırdığından değişme olasılığı yüksek olan birimlerin önceden belirlenmesine çalışılır.
- ...

Öğrenme tabanlı yöntemlerin oluşturulması (devamı)

2. Veri kümesinin oluşturulması

Eğitim kümesi (*training set*) ve sınama kümesi (*testing set*) oluşturulur.

- **Veri toplanacak olan projelerin seçilmesi**

Kurulan modelin kullanılacağı projelerle benzer özelliklere sahip referans projelerin seçilmesi uygun olur.

- Bilinen (koda ve raporlara sahip olunan) endüstriyel projeler
- Bilgilerine erişilebilen açık kaynaklı projeler

  Eğitim ve test amaçlı veri kümelerinin oluşturulabilmesi için yeterli sayıda sürümü ve hata raporları olan projeler tercih edilir.

  Son zamanlarda GitHub (https://github.com/) iyi bir kaynak oluşturmaktadır.
- Hazır veri kümeleri
  - Bug prediction dataset: http://bug.inf.usi.ch/index.php
  - Micro Interaction Metrics for Defect Prediction: http://lifove.github.io/mim/
  - Enhancing Change Prediction Models using Developer-Related Factors: http://tinyurl.com/y79ttbwa
  - PROMISE Software Engineering Repository: http://promise.site.uottawa.ca/SERepository/index.html
  - F. Arcelli Fontana, et.al. http://essere.disco.unimib.it/reverse/MLCSD.html

Öğrenme tabanlı yöntemlerin oluşturulması (devamı)

2. Veri kümesinin oluşturulması (devamı)

- **Toplanacak olan metriklerin seçilmesi**
  - Amaca (madde 1) uygun (ilişkisi olan) metriklerin seçilmesi
    Deneyimlerden ve önceki bilgilerden yararlanılarak uygun metrikler seçilebilir.
    Metrikler ile hedef (hata sayısı) arasında korelasyon aranabilir.
  - Başlangıçta çok sayıda metrik toplanıp daha sonra veri kümesi sadeleştirilebilir (madde 3).
  - Metrik türleri: Kod (tasarım) metrikleri, evrimsel metrikler, geliştirici temelli metrikler
  - Toplama maliyeti uygun olan metriklerin seçilmesi

2. Veri kümesinin oluşturulması (devamı)

- **Birimlerin (sınıf/metot) etiketlenmesi (*Labeling*)**
  Projede var olan birimler amaca uygun olarak etiketlenir ("bulunmak istenen kusur var"/ "kusur yok", "hataya yatkın"/ "hataya yatkın değil").
  - Kişi (uzman) tabanlı etiketleme:
    Proje ekibinin elemanları veya başka uzmanlar birimleri inceleyerek etiketlerler.
  - Otomatik etiketleme:
    Zaman kazanmak için, birimler belli algoritmalarla otomatik olarak etiketlenirler.
    Örneğin; yazılım projeleri devam ederken oluşturulan sürüm raporları incelenir; geçmişte "çok" hata çıkaran sınıflar "hata çıkarmaya yatkın" olarak etiketlenirler.
  - Yarı otomatik etiketleme:
    Kesin sonuç üretmeyen yöntemler ile aday birimler belirlenir ve bunlar uzmanlar tarafından tekrar gözden geçirilir.
    Örneğin sadece karmaşıklığı ve/veya bağımlılığı "çok yüksek" sınıflar incelenir.

2. Veri kümesinin oluşturulması (devamı)

Sonuçta ortaya matris şeklinde bir veri kümesi ortaya çıkar.

Satırlarda incelenecek olan yazılım birimleri (sınıflar, metotlar), sütunlarda ise metrikler yer alır.

En son sütun etiket bilgisini taşır.

Etiket çoğunlukla ikili değer alır (kusurlu/sağlıklı; değişime eğilimli / kararlı).

Metrikler
Farklı türlerde olabilir.
Kod, tarihsel, geliştirici

Birimler
Sınıf, metot, dosya

	M1	M2	M3	…	Mk	Etiket
S1						
S2						
S3						
…						
SN						

Öğrenme tabanlı yöntemlerin oluşturulması (devamı)

3. Veri Kümesinin Dengelenmesi ve Sadeleştirilmesi

- **Dengeleme:**

  Yazılım projelerinde birimler madde 2'de anlatıldığı gibi etiketlendiğinde çoğunlukla dengesiz veri kümeleri oluşmaktadır.

  Örneğin; sınıfların %90'ı kusursuz olarak işaretlenirken sade %10'u kusurlu olarak işaretlenmektedir.

  Dengesiz veri kümeleri öğrenme tabanlı yöntemlerin başarımını düşürmektedir.

  Kümeyi dengelemek için aşağıdaki yöntemler kullanılabilir:

  - "Kusursuz" etiketli sınıfları azaltma:
    - Projeye yeni eklenmiş olan sınıflar hata üretmemişlerse kümeden çıkartılırlar.
    - Geçmiş sürümlerde üzerinde az çalışılmış (değiştirilmemiş) sınıflar hata üretmemişlerse kümeden çıkartılırlar.
  - "Kusurlu" etiketli sınıfları artırma:
    - Geçmiş sürümlerde üzerinde az çalışılmış (değiştirilmemiş) sınıflar, hata üretmemiş olsalar bile belli metrik değerleri (karmaşıklık, bağımlılık gibi) belli eşik değerlerinin üstündeyse kümeye "kusurlu" olarak eklenirler.

3. **Veri Kümesinin Dengelenmesi ve Sadeleştirilmesi (devamı)**

- **Sadeleştirme:**

  Sadeleştirme iki şekilde olabilir: Gereksiz (yanıltıcı) sınıfların/metotların elenmesi, gereksiz niteliklerin (metriklerin) elenmesi.

- **Sınıfların / metotların elenmesi**
  - Projeye henüz yeni katılmış olan sınıflar, yeteri kadar değişmediği veya hata üretmediği için yeterli bilgiyi vermeyebilir veya yanıltıcı bilgi verebilirler.

    Örneğin, sınıf kusurlu olduğu halde henüz çok değişmediği için ürettiği hata sayısı da az olabilir.

    Sınıf aslında kusurlu olmadığı halde henüz olgunlaşmadığından üzerinde çok çalışıldığı için çok hata üretiyor olabilir.
  - Küçük birimler (çok kısa metotlar) yeterli bilgi taşımayabilir.
  - Düzeltilen (*refactoring*) birimler geçmişte çok sayıda hata üretmiş olsa bile güncel sürümlerde değiştiklerinden son metrik değerleri geçmişteki durumlarını yansıtmayacaktır.

  Bu tür sınıfların elenmesi öğrenme tabanlı sistemlerin başarımını artırmaktadır.

3. **Veri Kümesinin Dengelenmesi ve Sadeleştirilmesi (devamı)**

- **Sadeleştirme (devamı):**

- **Niteliklerin (metriklerin) azaltılması (*feature selection*)**

  Madde 2'de veri kümesi oluşturulurken eleme yapılmadan çok sayıda metrik toplandıysa iki sorun oluşabilir:
  - Bazı metriklerin etiketle bir ilişkisi olmayabilir (zayıf olabilir).
  - Bazı metrikler arasında yüksek korelasyon olabilir (aynı bilgiyi verirler).

  Gereksiz metriklerin (nitelikler) elenmesinin yararları:
  - Metrik toplama maliyetinin azalması
  - Oluşturulan yöntemin işlem gücü ve bellek gereksiniminin azalması
  - Oluşturulan modelin anlaşılır ve kolay yorumlanabilir olması
  - Bazı durumlarda yöntemin başarımının artması

- Sadeleştirme: Niteliklerin (metriklerin) azaltılması (***feature selection***) (devamı)

  Kullanılabilecek yöntemler:
  - **mRMR** (Minimum Redundancy Maximum Relevance): http://home.penglab.com/proj/mRMR/
  - **CFS** (Correlation-based Feature Selection for Machine Learning):
    Mark A. Hall, PhD Thesis, The University of Waikato, 1999.
    - Bu iki yöntem de etiketle yüksek korelasyonu olan ancak kendi aralarındaki korelasyon düşük olan nitelikleri seçmeye çalışır.
  - **PCA** (*Principal Component Analysis*): Temel Bileşen Analizi
    - PCA, metrikler ile etiketler arasındaki ilişkiyi dikkate almaz.
    - PCA, daha çok değişim içeren (varyansı yüksek) metriklerin seçilmesini sağlar.
    - Yöntem ayrıca aynı bilgiyi veren metriklerin de belirlenmesini sağlar. Böylece aynı bilgiyi taşıyan metriklerden sadece daha uygun olanı kümeye alınır.
    - Uygun metrikler, daha kolay toplanabilen ve yazılım takımı tarafından daha kolay yorumlanabilecek olan metriklerdir.
    - PCA yöntemi Bölüm 9'a ayrıca açıklanmıştır.
  - PCA, mRMR veya CFS ile birlikte de kullanılabilir.
    Örneğin CFS ile elde edilen metrikler arasında toplanması zor olanlar varsa bunlar PCA tarafından belirlenmiş olan benzerleriyle değiştirilebilirler.

Öğrenme tabanlı yöntemlerin oluşturulması (devamı)

4. Sınıflandırma (*classifier*) yöntemine karar verilmesi

En çok kullanılan sınıflandırma algoritmaları: Decision tree, Random Forest, Naïve Bayes

Projelere ve aranan kusurlara bağlı olarak farklı algoritmaların başarımları da farklı olmaktadır.

Bu nedenle uygun sınıflandırıcıyı bulmak için veri kümesi üzerinde denemeler yapmak gerekmektedir.

Sınıflandırıcı algoritmalarını hazır platformlarda veya programla dillerinin kütüphanelerinde bulmak mümkündür.

Örnekler:

- **Weka:** Hall M, Frank E, Holmes G (2009) The WEKA data mining software: an update. SIGKDD Explorations Newsletter 11(1): 10–18. doi:10.1145/1656274.1656278

  http://www.cs.waikato.ac.nz/~ml/weka/
- **Python** programlama dilini destekleyen makine öğrenmesi kütüphaneleri de yaygın olarak kullanılmaktadır.

  scikit-learn: https://scikit-learn.org/stable/

Öğrenme tabanlı yöntemlerin oluşturulması (devamı)

5. Oluşturulan yöntemin başarımının değerlendirilmesi

- Sınama (Test) kümesinin oluşturulması

Oluşturulan yöntemin başarısını ölçmek için de sınama amaçlı bir veri kümesi oluşturmak gereklidir.

Bu kümede de eğitim kümesinde olduğu gibi kusur aranan birimlerle ilgili metrikler ve etiketler yer alır.

Bura da etiketlemenin ikinci maddede anlatıldığı gibi uygun şekilde yapılması gereklidir.

Eğitim ve sınama kümeleri aşağıdaki gibi farklı kaynaklardan elde edilebilir.

- Model bir projenin belli bir kısmından elde edilen verilerle ile eğitilip aynı projenin diğer bölümlerindeki kusurları bulmak için kullanabilir.
- Model projenin ilk sürümlerinden elde edilen verilerle ile eğitilip aynı projenin sonraki sürümlerinde kusurları bulmak için kullanabilir.
- Model bir yazlım projesinden elde edilen veriler ile eğitilip başka yazılımlardaki kusurları bulmak için kullanılabilir.

5. Oluşturulan yöntemin başarımının değerlendirilmesi (devamı)

- Değerlendirme ölçüleri

Modelin ürettiği sonuçlar ile gerçek değerler karşılaştırıldığında aşağıdaki durumlar oluşabilir:

tp: *True positive*. Sınıfta kusur olduğu tahmin edildi. Sınıf gerçekte de kusurlu.

tn: *True negative*. Sınıfta kusur olmadığı tahmin edildi. Gerçekte de kusur yok.

fp: *False positive*. Sınıfta kusur olduğu tahmin edildi. Sınıfta gerçekte kusur yok.

fn: *False negative*. Sınıfta kusur olmadığı tahmin edildi. Sınıf gerçekte kusurlu.

Buna göre modelin başarımı aşağıdaki kriterlere göre değerlendirilebilir:

$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn}$$ Doğru tahminlerin tüm tahminlere oranı

$$Precision = \frac{tp}{tp + fp}$$ Kusurlu olduğu tahmin edilen sınıflardaki doğruluk oranı

$$Recall = \frac{tp}{tp + fn}$$ Bulunan kusurlu sınıfların sistemdeki tüm kusurlu sınıflara oranı. Tüm kusurlu sınıfları bulmadaki başarı.

$$F1 = 2\frac{Precision \cdot Recall}{Precision + Recall}$$ *Precision* ve *Recall* değerlerinin ağırlıklı ortalaması

Örnek Çalışma 1:

Sinan Eski, Feza Buzluca, "**Detection of God Classes in Object-Oriented Software Using Decision Tree**", Teknik Rapor, 2013.

Bu çalışmada, "God class" tasarım kusurunun belirlenmesi hedeflenmiştir.

Modeller başka kusurları belirlemek üzere de eğitilebilir.

Kullanılan metrikler:

Bu metrikler seçilirken deneyimlere dayanılarak "God class" kusuruyla ilgisi olabilecek metrikler seçilmiştir.

Bağımlılık (*Coupling*) Metrikleri:

- ATFD (Access to Foreign Data)
- CBO (Coupling Between Object Classes)
- RFC (Response For a Class)
- DIT (Depth of Inheritance Tree)

Uyum (*Cohesion*) Metrikleri:

- TCC (Tight Class Cohesion)
- LCOM (Lack of Cohesion in Methods)
- CAM (Cohesion Among Methods)

Karmaşıklık (*Complexity*) ve Boyut Metrikleri:

- WOC (Weight of Class)
- NOAM (Number of Accessor Methods)
- NOPA (Number of Public Attributes)
- WMC (Weighted Methods per Class)
- NOM (Number of Methods)
- LOC (Lines of Code)

Veri kümesinin oluşturulması:

- Kaynak projeler:

Bu çalışmada, veriler üç adet açık kaynaklı projeden elde edilmiştir:

ArgoUML, JFreeChart ve Xerces (Apache projesi)

- Yarı otomatik etiketleme:

Hangi sınıflarda kusur olduğunu belirlemek (etiketleme) için daha önce yayımlanmış olan 3 yöntemden yararlanılmıştır: E-Quality, DECOR, iPlasma.

Eğer üç yöntem de bir sınıfta kusur olduğunu gösteriyorsa o sınıf "kusurlu" kabul edilmiştir.

Eğer yöntemlerden ikisi bir sınıfı kusurlu olduğunu gösteriyorsa o sınıf gözle yeniden incelenerek son karar verilmiştir.

Sonuçta her sınıf için ilgili metrik değerlerini ve o sınıfta kusur olup olmadığı bilgisini (etiket) içeren bir veri kümesi oluşturulmuştur.

Her sınıf için (örneğin A) aşağıdaki gibi bir satır oluşturulmaktadır.

A	M1	M2	M3	M4	M5	…	M13	0/1

A sınıfına ilişkin metrik değerleri

Etiket:
kusur yok:0 / var:1

Sınıflandırma yöntemi (model) :

Bu çalışmada öğrenebilen karar ağacı modeli kullanılmıştır.

"God class" tasarım kusurunun belirlenmesi hedeflenmiştir.

Model başka kusurları belirlemek üzere de eğitilebilir.

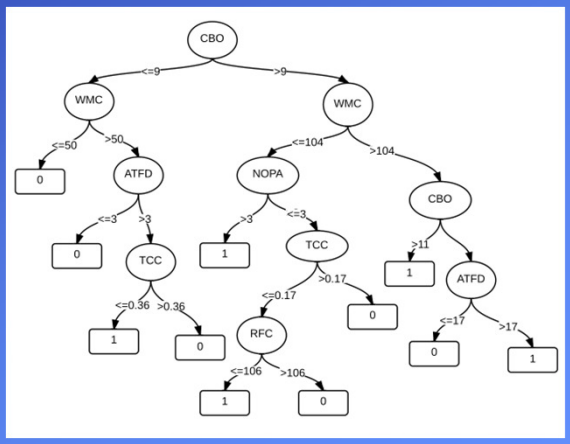
Karar ağacının yapısı:

Kök ve ara düğümlerde metrikler bulunur.

Dallarda metriklere uygulanacak kurallar (koşullar) yer alır. Koşulun sonucuna göre sağ ya da sol daldan devam edilir.

Yapraklarda sonuç (karar) yer alır. 0: Kusur yok, 1: Kusur var

C4.5 algoritmasının (Ross Quinlan) Weka programında bulunan gerçeklemesi J48 kullanılmıştır.



Modelin eğitilmesi (oluşturulması) ve sınanması:

Modeller üç farklı bakış açısıyla değerlendirilmiştir:

1. Modeli, bir programın belli bir kısmını kullanarak eğitirsek (oluşturursak) bu modeli aynı yazılımın diğer kısımlarındaki kusurları bulmak için kullanabilir miyiz?
2. Modeli, bir projenin ilk sürümlerinden elde ettiğimiz veriler ile eğitirsek bu modeli sonraki sürümlerde kusurları bulmak için kullanabilir miyiz?
3. Bir yazlım projesinden elde ettiğimiz veriler ile eğittiğimiz modeli başka yazılımlardaki kusurları bulmak için kullanabilir miyiz?

Deney Sonuçları:

1. Modeli bir programın belli bir kısmını kullanarak eğitirsek (oluşturursak) bu modeli aynı yazılımın diğer kısımlarındaki kusurları bulmak için kullanabilir miyiz?

   Model Apache Xerces projesi sürüm 2.7'ye uygulanmıştır.

   Bu projede 786 adet normal (kusursuz) ve 55 kusurlu (God Class) sınıf vardır.

   Eğitim kümesi: 470 normal, 35 kusurlu. Test kümesi: 316 normal, 20 kusurlu.

   Sonuçlar: tp=18, fn=2, fp=4, tn=312

	Precision	Recall	F-measure
God Class	0.818	0.900	0.857
Normal Sınıf	0.994	0.987	0.990

Önemli olan bu satırdır.

Deney Sonuçları (devamı):

2. Modeli bir projenin ilk sürümlerinden elde ettiğimiz veriler ile eğitirsek bu modeli sonraki sürümlerde kusurları bulmak için kullanabilir miyiz?

Model ArgoUML 0.24 ile eğitilmiş, ArgoUML 0.32 ile sınanmıştır.

Sonuçlar:

ArgoUML 0.32'de 1796 tane normal sınıf, 27 tane kusurlu sınıf vardır.

Yöntem 22 tane kusurlu sınıfı doğru olarak belirlemiştir.

Beş tane kusurlu sınıf yanlışlıkla normal olarak işaretlenmiştir.

İki tane normal sınıf yanlışlıkla kusurlu olarak işaretlenmiştir.

	Precision	Recall	F-measure
God Class	0.917	0.815	0.863
Normal Sınıf	0.997	0.999	0.998

Deney Sonuçları (devamı):

3. Modeli bir yazlım projesinden elde ettiğimiz veriler ile eğitirsek bu modeli başka yazılımlardaki kusurları bulmak için kullanabilir miyiz?

Model Xerces 2.7'den elde edilen veriler ile eğitilmiş, daha sonra JFreeChart 0.9.21 ve ArgoUML 0.32 ile sınanmıştır.

Sonuçlar:

JFreeChart 0.9.21:

	Precision	Recall	F-measure
God Class	0.897	0.875	0.886
Normal Sınıf	0.992	0.993	0.993

ArgoUML 0.32:

	Precision	Recall	F-measure
God Class	0.952	0.741	0.833
Normal Sınıf	0.996	0.999	0.997

Yöntem kullanılabilir düzeyde başarılı sonuçlar üretmektedir.

Örnek Çalışma 2:

Çağıl Biray, Feza Buzluca, "**A learning-based method for detecting defective classes in object-oriented systems",** IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2015.

http://dx.doi.org/10.1109/ICSTW.2015.7107477

Amaç: Yapısal (tasarımdan kaynaklanan) kusurlar nedeniyle hata çıkarmaya yatkın (*error-prone*) sınıfları bulmak.

Yararları:

- Kusurlu sınıfların sisteme eklenmesine izin verilmez.
- Kusurlu sınıflar proje ilerlemeden düzeltilerek yazılım geliştirmenin sonraki aşamalarında (testlerde) ve sürümlerinde zaman kazanılır.
- Hata çıkarma olasılığı yüksek olan sınıflar öncelikli olarak test edilir.

Yöntemin temel dayanak noktası:

- Kusurlu sınıflar; karmaşık, uyumsuz, çok bağımlı oldukları için bu tür sınıfları anlamak, değiştirmek, güncellemek, tekrar kullanmak zordur.
- Tasarımcılar ve kodlayıcılar bu tür sınıflar üzerinde çalışırken daha sık hata yaparlar. Bu nedenle yapısal kusurlara sahip olan sınıflar hata çıkarmaya yatkındırlar.
- Hata çıkarma olasılığı $\leftrightarrow$ yapısal kusur

Gözlemler ve Varsayımlar:

- Testlerde çıkan hataların çoğu kusurlu sınıflardan kaynaklanıyor.
- Yapısal kusuru olmayan sınıflar da bazen testlerdeki hata raporlarında yer alıyorlar.
- Kusurlu sınıflar genellikle esneklik ve uyumdan yoksun olduklarından bu sınıflarda bir güncelleme yapıldığında sonraki testlerde hata çıkartıyorlar.

  Güncelleme (değişiklik) $\rightarrow$ Hata

  Bir sınıf kusurlu da olsa eğer üzerinde bir işlem (değişiklik, ekleme) yapılmıyorsa testlerde hata çıkartmayabiliyor.
- Kusurlu sınıflar düzeltilmezse (*refactoring*) hata çıkartmaya devam ediyorlar.

Yöntemin Temeli:

- Endüstriyel bir projenin sürümleri ve ilgili hata raporları incelendi.
  - Çok değişen ve sık hata üreten sınıflar "kusurlu" olarak işaretlendi.
  - Değişen ama az hata üreten sınıflar "sağlıklı" olarak işaretlendi.
  - Hiç değişmeyen sınıflar ayrıca metrikler ile incelendi.
- Sınıfların seçilen metrikleri ile bir eğitim kümesi oluşturuldu.
- Veri kümesi ile eğitilerek bir karar ağacı modeli oluşturuldu.

Modelin Oluşturulması:

Eğitim amaçlı veri kümesinin oluşturulması:

- Telekom firmaları için geliştirilen endüstriyel bir yazılımın x adet sürümü ve raporları incelendi. Bu çalışmada x = 46.
- Sınıfları etiketlemek için (kusurlu/sağlıklı) çok değişen ve sık hata çıkaran sınıflar belirlenmeye çalışıldı.

Hesaplanan parametreler:

- **Hata sayısı ErrC (*Error Count*):** İncelenen x sürümde c sınıfında çıkan hata sayısı.

$ErrC_C = \sum_{i=1}^{x} e_{c,i}$

$e_{c,i} = 1$ eğer c sınıfında i. sürümde hata düzeltme yapıldıysa

$e_{c,i} = 0$ eğer c sınıfında i. sürümde hata çıkmadıysa

- **Değişim isteği sayısı CRC (*Change Request Count*):** İncelenen x sürümde c sınıfında müşteri isteği (CR) nedeniyle yapılan değişim sayısı.

$CRC_C = \sum_{i=1}^{x} r_{c,i}$

$r_{c,i} = 1$ eğer c sınıfı i. sürümde CR nedeniyle değiştiyse

$r_{c,i} = 0$ eğer c sınıfı i. sürümde CR nedeniyle değişmediyse

Hesaplanan parametreler (devamı):

- **Toplam değişim sayısı ChC (*Change Count*):** İncelenen x sürümde c sınıfında meydana gelen toplam değişim sayısı.

$ChC_C = ErrC_C + CRC_C$

- **Hata sıklığı EF (*Error Frequency*):** Değişim başına düşen hata sayısı

$EF_C = \frac{ErrC_C}{ChC_C}$

Bu değerin (EFc) yüksek olması o sınıfta (c) yapısal kusur olduğunun işaretidir.

Sınıfların Etiketlenmesi:

- Proje ekibiyle birlikte sınıflar incelendiğinde kusurlu sınıfların değişim sayıları ve hata sıklıkları için iki **eşik değeri** belirlendi.

ChC için t_1, EF için t_2

- Çok değişen ve sık hata üreten sınıflar "kusurlu" olarak etiketlendi.

tag_C = Defective, if (ChCc ≥ t1 and EFc ≥ t2)

Sınıfların Etiketlenmesi (devamı):

- Hiç değişmeyen veya az değişen sınıfları gerçekçi biçimde etiketlemek için ek bir incelemeye gerek vardır.
- Bu sınıflar sağlıklı olabilir veya tesadüfen incelenen sürümlerde hiç değişime uğramadıkları için yapısal hatalarını gizliyor olabilirler.
- Bu tür sınıfların sadece karmaşıklık (WMC) metrikleri incelenerek ortalamanın 1,5 katı üstünde olanlar da "kusurlu" olarak işaretlendi.

$$tag_c = \begin{cases} Defective,\ if\ (ChC_c \geq t_1\ and\ EF_c \geq t_2), \\ Defective,\ if\ ((ChC_c < t_1\ or\ EF_c < t_2)\ and\ \ WMC_c \geq AVG*1.5), \\ Healthy,\ otherwise. \end{cases}$$

t_1: Değişim sayısı eşik değeri

t_2: Hata oranı eşik değeri

- Bu çalışmada projenin yazılım ekibiyle birlikte incelenmesi sonucunda eşik değerleri aşağıdaki gibi seçilmiştir.

$t_1 = 5$, $t_2 = 0.25$

Kullanılan Metrikler:

- Yapısal kusurlarla ilgili olabilecek farklı türlerde 23 adet metrik toplanmıştır.
- Metriklerin türleri: Karmaşıklık ve boyut, bağımlılık, uyum, arayüz ve kalıtım.

Metriklerin tam listesi ve ayrıntıları için ilgili yazıya bakınız.

Eğitim kümesi:

- Sonuçta her sınıf için ilgili metrik değerlerini ve o sınıfta kusur olup olmadığı bilgisini içeren bir veri kümesi oluşturulmuştur.
- Küme bir matris şeklindedir ve her sınıf için (örneğin A ve B sınıfları) aşağıdaki gibi bir satır oluşturulmaktadır.

	Sınıflara ilişkin metrik değerleri							kusur yok/var
A	M1	M2	M3	M4	M5	…	M23	0/1
B	M1	M2	M3	M4	M5	…	M23	0/1

- İncelenen projeden 247 satırlı bir küme elde edilmiştir.
- Kusurlu sınıf sayısı: 47
- Sağlıklı sınıf sayısı: 200
- Yazılım projelerinde bu sayılar genellikle dengesizdir ve bu durum öğrenme tabanlı sistemlerin başarımını düşürmektedir.

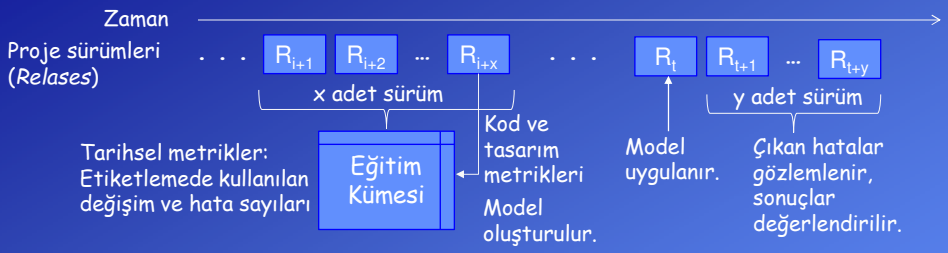
Karar Ağacı:

Veri kümesi kullanılarak Weka'daki J48 algoritması ile karar ağacı oluşturulmuştur.

Bu algoritma kümedeki metrikler arasında eleme yaparak sadece 5 tanesini seçmiştir:

- CBO (Coupling between Object classes)
- LCOM (Lack of Cohesion in Methods)
- WOC (Weight of a Class): Bir sınıfın ara yüzünde yer alan (*public*), sırf erişim amaçlı olmayan (*non-accessor*) metotların sınıftaki tüm açık (*public*) metotlara oranı.
- HIT (Height of Inheritance Tree): Sınıfın kalıtım ağacındaki yüksekliği (alt sınıflara olan uzaklığı)
- NOM (Number of Methods)

Yöntemin Sınanması (Testler):



- Projenin x adet sürümünden elde edilen tarihsel veriler ve R_{i+x} sürümünden elde edilen tasarım metrikleri ile model (karar ağacı) oluşturuldu.
- Projenin güncel (R_t) sürümüne model uygulanarak kusurlu sınıflar belirlendi.
- Bundan sonra gelen y adet sürümde (gözlem sürümleri) sınıfların değişme sayıları ve hata çıkarma sıklıkları incelendi.

  Bu aşamada, kusurlu sınıflar belirlenirken eğitim kümesi oluşturulurken kullanılan aynı eşik değerleri kullanıldı.
- Test sürümünde (R_t) kusurlu olarak belirlenen sınıflar daha sonraki sürümlerde (gözlem sürümleri) gerçekten çok hata (eşik değerine göre) çıkartıyorlar mı?

  Sağlıklı olarak işaretlenen sınıflar hatasız mı?

Deneysel Sonuçlar:

Proje A:

- Eğitim kümesi bu projeden elde edilen veriler ile oluşturulmuştur.
- Oluşturulan karar ağacı eğitimde kullanılan sürümlerden sonraki sürümler üzerinde sınanmıştır.
- Sık hata üreten sınıflardan %81'i belirlendi.
- Hiç değişmeyen (hata oranı = 0/0) 18 adet sınıf kusurlu olarak işaretlendi.

  Bu sınıflar yazılım ekibi tarafından incelendikten sonra 13 tanesinin gerçekten kusurlu olduğu belirlendi.

ErrC / ChC = EF	Toplam kusurlu sınıf sayısı	Doğru olarak belirlenen sınıf sayısı
8 / 11 = 0.73	1	1
7 / 11 = 0.64	1	0
6 / 12 = 0.5	1	1
6 / 10 = 0.6	1	1
6 / 7 = 0.86	1	1
5 / 11 = 0.45	1	1
5 / 10 = 0.5	1	1
5 / 9 = 0.56	1	0
5 / 8 = 0.63	1	1
5 / 7 = 0.71	1	1
4 / 10 = 0.4	1	1
4 / 6 = 0.67	2	0
4 / 5 = 0.8	1	1
3 / 7 = 0.43	2	2
3 / 6 = 0.5	1	1
3 / 5 = 0.6	2	2
2 / 5 = 0.4	2	2

Deneysel Sonuçlar (devamı):

Proje B:

- Proje A'dan elde edilen eğitim kümesi ile oluşturulan karar ağacı Proje B'ye uygulanarak kusurlu sınıflar belirlenmiştir.
- Sık hata üreten sınıflardan %83'ü belirlendi.
- Hiç değişmeyen (hata oranı = 0/0) 7 adet sınıf kusurlu olarak işaretlendi.

  Bu sınıflar yazılım ekibi tarafından incelendikten sonra 4 tanesinin gerçekten kusurlu olduğu belirlendi.

ErrC / ChC = EF	Toplam kusurlu sınıf sayısı	Doğru olarak belirlenen sınıf sayısı
10 / 10 = 1	1	1
9 / 11 = 0.82	1	1
8 / 9 = 0.89	1	1
7 / 7 = 1	1	1
6 / 8 = 0.75	1	1
6 / 7 = 0.86	1	0
5 / 6 = 0.83	1	1
5 / 5 = 1	1	1
4 / 5 = 0.8	2	2
3 / 6 = 0.5	1	0
3 / 5 = 0.6	1	1

Örnek Çalışma 3

Fikret Aktaş, "Nesneye Yönelik Sistemlerde Kusurlu Sınıfların Öngörülmesi İçin Makine Öğrenmesi Temelli Bir Yöntem Oluşturulması", Yüksek Lisans Tezi, İTÜ Fen Bilimleri Enstitüsü, Nisan 2018.

Fikret Aktaş, Feza Buzluca, "A Learning-Based Bug Prediction Method for Object-Oriented Systems", 17th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2018), June 6-8, 2018, Singapore.
https://doi.org/10.1109/ICIS.2018.8466535

Amaç:

- Hata çıkarmaya yatkın sınıfları bulmak.
- Anlaşılır bir model oluşturup hatalara neden olan kusurları belirlemek.

Çalışmanın Temel Noktaları:

- Tasarım/kod, tarihsel (*historical*), etkileşim (*micro interaction*) metrikleri birlikte kullanılmıştır.
- Veri kümesi oluşturulurken dengelemeye dikkat edilmiştir.
- CFS ve PCA ile küme sadeleştirilmiştir.
- Sınıflandırma için Random Forest ve karar ağaçları kullanılmıştır.

Örnek Çalışma 3 (devamı)

Veri Kümesi:

- Ham metrik kümesi olarak olarak hazır veriler kullanılmıştır.
  Taek Lee, et al. "Developer Micro Interaction Metrics for Software Defect Prediction", IEEE Transactions on Software Engineering, 2016.
  http://lifove.github.io/mim/
- Eclipse Bugzilla, Aralık 2005- Haziran 2010 arasında çeşitli projelerle ilgili bilgiler toplanmış.
  Projeler: Mylyn, Team, JDT, Etc (birkaç projenin karışımı)

Toplam 3077 Sınıf			
Mylyn 1084 Sınıf	Team 364 Sınıf	JDT 244 Sınıf	Etc 1385 Sınıf

- Her sınıf için farklı tiplerde olmak üzere toplam 81 metrik toplanmıştır.
- 15 tarihsel metrik (HM), 42 kod metriği (CM), 24 mikro etkileşim metriği (MIM)

Etiketleme:

- Geçmiş sürümlerde **sık** hata çıkaran sınıflar "kusurlu" olarak işaretlenmiştir.
- Burada sınıfın toplam hata sayısı değil, değişim (revizyon) başına ürettiği hata sayısı dikkate alınmıştır.

Label(s) = "Buggy" if (#ofBugfix$_Z$(s) / #ofRevision$_Z$(s)) > E

  - Label(s): s sınıfının etiketi "Buggy" / "Clean"
  - #ofBugfix$_Z$(s): s sınıfının Z zaman penceresi içinde ürettiği hata sayısı
  - #ofRevision$_Z$(s): s sınıfının Z zaman penceresi içinde uğradığı değişiklik sayısı
  - E: Proje için belirlenen hata sıklığı eşik değeri
  - Z: Zaman penceresi (*time window*), ilgili projenin ne kadar süreyle gözlemlendiği, kaç sürümden veri alındığını gösterir.

    Gözlem süresi projeye göre uygun seçilmelidir.

    - Kısa gözlem süresi: Sınıfların hata / değişim karakterini ortaya çıkaracak yeterli veri toplanamayabilir.

    - Uzun gözlem süresi: Geçen süre içinde değişiklikler ve düzeltmeler nedeniyle sınıfların karakterleri değişir.

    Henüz başlangıç aşamasında olan veya olgunlaşmış olan projelerde bu süre farklı şekilde seçilmelidir.

Etiketleme: Eşik değerinin seçimi

Ne kadar sık hata üreten sınıflar kusurlu olarak işaretlenecek?

İki unsur belirleyici olmaktadır.

1. Veri kümesini dengeli yapmak

   Kusurlu ve kusursuz sınıfların sayısı birbirine yakınsa makine öğrenmesi yöntemlerinin başarımı artmaktadır.

2. Kusurlu sınıf sayısını belli bir sayının altında tutmak

   Oluşturulan yöntem, incelenen projedeki çok sayıda sınıfı kusurlu olarak belirlerse, projenin sınama ve düzelteme maliyetleri yükselir.

- Küçük eşik değeri:
  - Çok sayıda sınıf kusurlu olarak işaretlenir. Gerçekten kusurlu oldukları şüphelidir.
  - Yöntem çok sayıda kusurlu sınıf bulur; sınama ve düzeltme maliyetleri artar.
  - Küme dengeli olabilir.
- Büyük eşik değeri:
  - Az sayıda sınıf kusurlu olarak işaretlenir. Kusurlu olma olasılıkları yüksektir.
  - Yöntem az sayıda kusurlu sınıfı bulur; sınama ve düzeltme maliyetleri düşer.
  - Küme dengesiz olduğu için öğrenme tabanlı yöntemin başarımı düşebilir.

Etiketleme: Eşik değerinin seçimi (devamı)

- Örnek projelerde, farklı eşik değerleri ile etiketlemeler yapılarak oluşturulan veri kümelerine, henüz nitelik seçimi (*feature selection*) yapılmadan Random Forest yöntemi uygulanmıştır.
- Kusurlu sınıf sayısının, projedeki toplam sınıf sayısına oranı yaklaşık olarak %30 civarında tutulursa hem yöntemin başarımı hem de test maliyetleri açısından uygun sonuçlar elde edilebileceği gözlemlenmiştir.

Etc Projesi Eşik Değeri Karşılaştırmaları:

Eşik Değeri E	2	1.5	1.25	1	0.9	0.8
Kusurlu/Toplam	0.05	0.13	0.2	0.28	0.35	0.43
F-Ölçütü	0.118	0.245	0.358	0.511	0.672	0.732
Recall	0.063	0.145	0.235	0.396	0.612	0.729
Precision	1	0.794	0.753	0.719	0.746	0.745

Mylyn Projesi Eşik Değeri Karşılaştırmaları:

Eşik Değeri E	2.9	2.7	2.5	2.4	2.2	2.0
Kusurlu/Toplam	0.1	0.17	0.25	0.35	0.48	0.58
F-Ölçütü	0.027	0.127	0.371	0.588	0.724	0.675
Recall	0.027	0.073	0.265	0.527	0.669	0.578
Precision	0.5	0.5	0.613	0.666	0.79	0.811

Etiketleme: Eşik değerinin seçimi (devamı)

JDT Projesi Eşik Değeri Karşılaştırmaları:

Eşik Değeri E	2	1.75	1.5	1.25	1	0.9
Kusurlu/Toplam	0.09	0.176	0.25	0.34	0.426	0.5
F-Ölçütü	0.167	0.393	0.741	0.783	0.831	0.843
Recall	0,091	0.279	0.656	0.783	0.875	0.818
Precision	1	0.667	0.851	0.783	0.791	0.868

Team Projesi Eşik Değeri Karşılaştırmaları:

Eşik Değeri E	1	0.9	0.8	0.6	0.4	0.3
Kusurlu/Toplam	0.15	0.21	0.26	0.32	0.45	0.52
F-Ölçütü	0.282	0.462	0.604	0.783	0.831	0.808
Recall	0.179	0.346	0.531	0.75	0.818	0.82
Precision	0.667	0.692	0.699	0.818	0.844	0.797

- Örnek projelerde bir sürümde birden fazla hata düzeltilebildiği için E>1 dir.
- Her proje için farklı bir eşik değeri E uygun olmaktadır.
- Kusurlu/Toplam sınıf sayısı oranı tüm projelerde yaklaşık olan %30'dur.

Veri kümesini dengeleme ve sadeleştirme:

Dengeleme:

- Eşik değeri seçilirken kusurlu/toplam sınıf sayısı oranı (~%30) dikkate alınmıştır.
- Ek olarak; belli bir sayıdan daha az değişime uğramış olan sınıflar eğer hata üretmemişlerse veri kümesinden çıkarılmıştır.

  Neden: Değişime uğramayan sınıflar yapıları kusurlu olsa bile hata üretmeyebilirler.

  Bu çalışmada $\#ofRevision_Z(s) < 5$ olan sınıflar veri kümesinden çıkarılmıştır.
- Dengeleme tüm projelerde başarımı artırmıştır.

  Hangi sınıfların eleneceği zaman penceresinin uzunluğuna bağlı olarak belirlenebilir.

  Örneğin; Z = 40 sürüm ise eleme için kullanılacak eşik değeri bunun 8-10'da biri, 5 veya 4 olabilir.

Sadeleştirme (nitelik seçimi):

A) Elde edilen veri kümelerine **CFS**[1] (Correlation-based Feature Selection for Machine Learning) uygulanarak etiketlerle korelasyonu en yüksek olan metrikler seçilmiştir.

Nitelik sayısı 81'den yaklaşık olarak 5-10 arasına indirilmiştir.

Örnek olarak Etc E = 0.9 kümesi için CFS aşağıdaki metrikleri seçmiştir.

- NumInterruptions: Sınıfı yazan kişi belli bir sürede kaç kere ara veriyor.
- TimeSinceLastTask: Sınıf üzerindeki son çalışmadan bu yana geçen süre
- AvgLineCode: Sınıfın kod uzunluğu
- AvgOfAddedLines: Sınıfa eklenen satır sayısının ortalaması
- Age: Sınıfın yaşı [hafta]

[1]Mark A. Hall, Correlation Based Feature Selection for Machine Learning, PhD Thesis, Department of Computer Science, University of Waikato, 1999.

Sadeleştirme (nitelik seçimi) (devamı):

B) PCA yöntemi kullanılarak aynı temel bileşende (*principal component*) yer alan metrikler belirlenmiştir.

- Aynı temel bileşende yer alan metriklerin benzer bilgiler verdiği düşüncesiyle CFS'in seçtiği bazı metrikler değiştirilmiştir.
- Yazılım ekibine hataların kaynağı ile ilgili bilgi verebilecek metrikler tercih edilmiştir.

  Örneğin AvgOfAddedLines yerine SumEssential metriği kullanılmıştır.
  - AddedLines metriğini sebep sonuç ilişkisi açısından yorumlamak zordur.
  - Sınıfa çok satır eklendiği için mi sık hata çıkarmaktadır (kusurludur)?
  - Sınıf sık hata çıkardığı için mi düzeltmeler sırasında çok satır eklenmektedir?

Sonuçlar:

Oluşturulan veri kümeleri üzerinde 10 katlı çaprazlama yöntemiyle kusurlu sınıflar belirlenmeye çalışılmıştır.

ETC kümesi için elde edilen sonuçlar:

Eşik Değeri E = 0.9	Tüm Metrikler	Sadeleştirilmiş Metrik Seti	Sadeleştirilmiş ve Dengelenmiş Set
F-Ölçütü	0.672	0.637	0.667
Recall	0.612	0.566	0.624
Precision	0.746	0.728	0.716

Tüm veri kümelerinin birleşiminden oluşan ALL kümesi için elde edilen sonuçlar:

Eşik Değeri E = 1.75	Tüm Metrikler	Sadeleştirilmiş Metrik Seti	Sadeleştirilmiş ve Dengelenmiş Set
F-Ölçütü	0.793	0.773	0.79
Recall	0.738	0.725	0.758
Precision	0.858	0.828	0.824

- Dengeleme ve gereksiz sınıfların elenmesi başarımı arttırmıştır.
- Metrikleri azaltmak ve değiştirmek ise (CFS + PCA) başarımı çok az düşürmüştür.
- Az metrik ile çalışmak hem yöntemin oluşturma ve çalıştırma maliyetini düşürmekte hem de ortaya anlaşılabilir bir model çıkmasını sağlamaktadır.