T.C.

BALIKESİR ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Kaynak Kod Benzerliğini Bulanık Mantık Yaklaşımıyla Tespit Eden Sistem

202013709019- Ahmet Eren ÖZTÜRK

BMM4101 YAPAY ZEKA TEKNİKLERİ FİNAL ÖDEVİ

Danışman: Dr. Öğr. Üyesi Kadriye ERGÜN

BALIKESİR, Ocak – 2024

İÇİNDEKİLER

1. GİRİŞ

1.1 Çalışmanın Tanıtımı ve Amacı

Bu sistem, Python programlama dilinde yazılan kaynak kodların benzerlik analizini yapmak için Abstract Syntax Tree (AST) analizi ve metinsel içerik karşılaştırmasını birleştiren bir yöntem önermektedir. AST analizi, kodun yapısal özelliklerini incelerken, metinsel karşılaştırma(difflib) ise kodun yazılı içeriğine odaklanır. Her iki ölçüm de, genel bir benzerlik puanı üretmek için bir bulanık mantık modeline entegre edilmiştir.

Proje, Python programlama dilinde yazılmış kaynak kodların benzerliğini değerlendirirken pratik ve erişilebilir bir yaklaşım sunmayı amaçlamaktadır. Bulanık mantık modeli, AST ve metinsel benzerlik ölçümlerinden elde edilen verileri dengeli bir şekilde birleştirerek, kodların benzerlik derecesini belirlemeye yardımcı olur. Bu sistem, yazılım geliştirme ekiplerine ve akademik kurumlara, Python kodlarının veya yazılım ürünlerinin orijinalliğini değerlendirmede yardımcı bir araç olarak hizmet edebilir. Ayrıca, telif hakkı ihlallerini önleme ve özgünlüğü teşvik etme konusunda da faydalı olabilir.

Projenin, Python programlama dilinde yazılan kaynak kod benzerliği konusunda kesin sonuçlar sunmaktan ziyade, bir başlangıç noktası ve yardımcı bir rehber olarak görülmesi amaçlanmıştır. Sistem, özellikle öğretim üyeleri ve yazılım geliştiriciler için, Python kod incelemelerinde ve ödev değerlendirmelerinde kullanılabilir bir referans sağlar. Ancak, sonuçların her zaman uzman bir incelemeyle desteklenmesi gerektiğinin altı çizilmelidir.

Anahtar Kelimeler: Kaynak Kod Analizi, Bulanık Mantık, Abstract Syntax Tree, Metinsel İçerik Karşılaştırması

1.2 Kullanılan Programlar ve Yazılımlar

Bu projede gerçekleştirilen tüm kodlamalar Kaggle Notebook üzerinde yapılmıştır. Kaggle'ın sağlamış olduğu esnek ve güçlü kodlama ortamı, projenin ihtiyaç duyduğu çeşitli kütüphaneleri ve araçları entegre etmeyi kolaylaştırmıştır. Özellikle, kaynak kodların Abstract Syntax Tree (AST) analizinde Python'un 'ast' kütüphanesi, metinsel içerik karşılaştırmasında 'difflib' modülü, veri işleme ve matematiksel hesaplamalarda 'numpy', ve bulanık mantık modellerinin oluşturulması ve simülasyonunda 'skfuzzy' kütüphanesi kullanılmıştır. Bu araçların her biri, projenin farklı aşamalarında kritik roller oynamış ve kaynak kod benzerliği analizinin etkin bir şekilde gerçekleştirilmesini sağlamıştır.

1.2.1 Abstract Syntax Tree (AST)

Abstract Syntax Tree, kaynak kodun yapısal bir temsilidir ve programlama dillerinde semantik analizin temel bir bileşenidir. Bir AST, kodun soyut bir ağaç yapısında görselleştirilmesini sağlar, burada her düğüm kodun bir parçasını (örneğin, ifadeler, değişkenler, işlemler) temsil eder. AST, kodun yazımından ziyade mantığını ve yapısını vurgular.

AST, derleyiciler ve yorumlayıcılar tarafından geniş ölçüde kullanılır. Kod optimizasyonu, hata analizi, ve refaktörleme gibi birçok programlama aracı ve teknik için de önem taşır. Ayrıca, statik kod analizi, güvenlik incelemeleri ve hatta otomatik kod üretimi gibi alanlarda da kullanımı bulunmaktadır.

Bu projede AST kullanımının temel gerekçesi, kaynak kodun yapısal özelliklerini analiz ederek, iki farklı kod parçası arasındaki yapısal benzerlikleri belirlemektir. AST'nin bu kullanımı, kodun sadece metinsel içeriğini değil, aynı zamanda kodun nasıl yapılandırıldığını ve hangi programlama yapılarının kullanıldığını da değerlendirmemize olanak tanır.

Projede, Python'un 'ast' modülü kullanılarak kaynak kodlardan AST'ler oluşturulur. Bu AST'ler, daha sonra düğüm türleri, yapılar ve içerikler açısından incelenir. AST analizi, iki kod parçasının benzerlik düzeyini belirlemek için kullanılır. Örneğin, iki kod parçası arasındaki düğüm sayısı farkı, yapısal benzerliklerin ve farklılıkların bir göstergesi olarak değerlendirilir. Bu yöntem, özellikle farklı kod parçalarının temel yapısal unsurlarını ve bu unsurlar arasındaki ilişkileri anlamada oldukça etkilidir.

1.2.2 Difflib

Difflib, Python programlama dilinin standart kütüphanesinin bir parçası olan, metinsel veriler arasında farkları ve benzerlikleri tespit etmek için kullanılan bir modüldür. Bu modül, özellikle metin karşılaştırma ve analiz işlemlerinde geniş kullanım alanına sahiptir. Difflib, metin blokları arasındaki farklılıkları ve benzerlikleri belirlemek için *SequenceMatcher* adında bir sınıf sağlar. Bu sınıf, iki metin arasında ne kadar "yakın" veya "uzak" olduklarını belirleyen oransal bir skor üretir. Bu özellik, kod incelemeleri, metin düzenlemeleri, belge karşılaştırmaları ve hatta intihal kontrolü gibi çeşitli uygulamalarda kullanılır.

Bu projede Difflib modülünün temel kullanımı, iki kaynak kod parçası arasındaki metinsel benzerliği değerlendirmektir. Bu, kodların yazılı içeriğinin doğrudan karşılaştırılması anlamına gelir. Proje kapsamında, Difflib'in *SequenceMatcher* sınıfı kullanılarak, iki farklı kod bloğu arasında ne kadar benzerlik olduğu hesaplanır. Bu benzerlik skoru, yüzdelik bir değer olarak ifade edilir ve iki kod parçasının metinsel olarak ne kadar benzer olduğunu gösterir.

Bu yaklaşımın projedeki önemi, kaynak kodların sadece yapısal özelliklerine değil, aynı zamanda doğrudan yazılı içeriğine de odaklanmaktır. Bu, özellikle kodların kelime ve ifade düzeyindeki benzerliklerini belirlemede etkilidir. Örneğin, iki farklı fonksiyonun benzer değişken adları, yorum satırları veya kod blokları kullanıp kullanmadığını tespit etmek için bu yöntem kullanılabilir.

Projede Difflib'in kullanımı, AST analiziyle birlikte, kaynak kod benzerliğinin daha kapsamlı bir değerlendirmesine olanak tanır. Metinsel benzerlikler, kodun yüzey düzeyindeki benzerliklerini ortaya çıkarırken, AST analizi kodun daha derin yapısal özelliklerine odaklanır. Bu iki yöntemin kombinasyonu, kaynak kod benzerliğinin daha detaylı ve kapsamlı 0bir şekilde analiz edilmesini sağlar.

1.2.3 NumPy

NumPy, özellikle bilimsel hesaplamalar için tasarlanmış, Python programlama dilinin popüler bir kütüphanesidir. Çok boyutlu diziler ve matrisler üzerinde yüksek performanslı işlemler yapabilme kapasitesiyle tanınır. NumPy, büyük veri kümeleri üzerinde verimli ve hızlı hesaplamalar yapmak için gelişmiş matematiksel fonksiyonlar ve işlevsellikler sağlar. Veri analizi, lineer cebir, Fourier dönüşümleri ve rastgele sayı üretimi gibi birçok alanda geniş kullanım alanına sahiptir. Bu kütüphane, veri bilimi, makine öğrenmesi, yapay zeka ve çok daha fazlası gibi alanlarda yaygın olarak tercih edilmektedir.

Bu Projede NumPy'ın kullanımı, özellikle veri manipülasyonu ve matematiksel hesaplamalar için olmuştur. NumPy, projede, kaynak kod benzerliği analizinde elde edilen verilerin işlenmesi ve analiz edilmesi sırasında kullanılmıştır. Özellikle, bulanık mantık modelinin kurulumunda ve çalıştırılmasında NumPy'den yararlanılmıştır.

Bulanık mantık modeli, kaynak kodlar arasındaki benzerliği belirlemek için çeşitli girdi ve çıktı değişkenlerini kullanır. Bu değişkenlerin tanımlanması, işlenmesi ve modelin hesaplamalarında NumPy kütüphanesinin sağladığı işlevsellikler devreye girer. NumPy'ın sağladığı veri yapıları ve matematiksel işlevler, modelin daha verimli ve hatasız bir şekilde çalışmasını sağlar. Ayrıca, AST analizi ve metinsel benzerlik hesaplamalarında elde edilen verilerin işlenmesi ve analiz edilmesi sürecinde de NumPy rol oynamaktadır.

1.2.4 Skfuzzy

Skfuzzy, Python için geliştirilmiş bir kütüphanedir ve bulanık mantık algoritmalarını ve sistemlerini uygulamak için kullanılır. Bulanık mantık, geleneksel ikili mantık sistemlerinin aksine, gerçek dünyanın daha nüanslı ve belirsiz yönlerini modelleyebilir. Bu yaklaşım, her şeyin kesin "doğru" veya "yanlış" olmadığı, bunun yerine çeşitli "doğruluk dereceleri"

içerebileceği durumlar için uygundur. Skfuzzy, bulanık setler, bulanık sayılar, bulanık işlemler ve bulanık kontrol sistemleri gibi konseptleri destekler. Bu kütüphane, özellikle sistem kontrolü, karar verme süreçleri ve karmaşık problem çözme gibi alanlarda geniş kullanım alanına sahiptir.

Projede Skfuzzy, kaynak kodlar arasındaki benzerlikleri değerlendiren bulanık mantık modelinin oluşturulması ve simülasyonu için kullanılmıştır. Bu projede, AST analizi ve metinsel benzerlik analizi sonuçları, bulanık mantık modeline girdi olarak verilmiştir. Bu model, her iki türden elde edilen verileri birleştirerek, iki kod parçası arasındaki genel benzerlik derecesini hesaplamak için kullanılmıştır.

Skfuzzy kütüphanesi, projedeki bulanık mantık modelinin temelini oluşturur. Bu model, girdiler olarak AST ve metinsel benzerlik(difflib) skorlarını alır ve bu skorları, kod parçaları arasındaki genel benzerliği belirlemek için kullanılan bulanık mantık kuralları ile işler. Model, belirlenen bu kurallara göre çıktı üretir ve sonuç olarak, iki kod parçası arasındaki genel benzerlik skorunu verir.

1.3 Measure of Software Similarity (MOSS)

MOSS'un Tanımı ve Genel Kullanımı:

Measure of Software Similarity (MOSS), Stanford University tarafından geliştirilen ve akademik çevrelerde geniş çapta kullanılan bir yazılım benzerliği tespit aracıdır. MOSS, özellikle eğitim alanında öğrencilerin gönderdikleri programlama ödevlerinin özgünlüğünü kontrol etmek amacıyla tasarlanmıştır. Geniş bir yelpazede programlama dillerini destekleyen bu araç, kaynak kodlar arasındaki yapısal ve semantik benzerlikleri belirlemek için gelişmiş algoritmalar kullanır. MOSS, intihal veya telif hakkı ihlallerini tespit etmeyi ve böylece akademik dürüstlüğü korumayı hedefler.

MOSS'un Çalışma Prensibi ve Özellikleri:

MOSS, kaynak kodları analiz ederken, yalnızca metinsel benzerliklere odaklanmaz; kodun yapısal ve semantik özelliklerini de dikkate alır. Bu, MOSS'un kelime bazında benzerliklerin yanı sıra algoritmik ve fonksiyonel benzerlikleri de tespit edebilmesini sağlar. MOSS, kod parçalarını karşılaştırırken, yorum satırları, boşluklar ve diğer görsel öğeler gibi alakasız detayları göz ardı ederek kodun mantıksal yapısına odaklanır.

MOSS, öncelikle "**winnowing**" adı verilen bir algoritma kullanır. Bu algoritma, kod içerisinden belirli "n-gram" dizilerini seçer ve bu dizilerin karmaşık **hash** değerlerini hesaplar. Bu yöntem, kodun özgün parçalarını ve genel yapısını belirlemeye yardımcı olur. Ardından, bu hash değerleri karşılaştırılarak, iki kod arasındaki yapısal benzerlikler tespit edilir.

MOSS'un Uygulama Alanları:

MOSS, özellikle üniversitelerde ve eğitim kurumlarında yaygın olarak kullanılmaktadır. Öğrencilerin programlama dilleri üzerine yaptıkları ödevlerin özgünlüğünü kontrol etmek için ideal bir araçtır. MOSS'un sağladığı raporlar, öğretim üyelerine öğrencilerin işlerindeki benzerlik derecesini ve bu benzerliklerin potansiyel kaynaklarını gösterir. Bu, öğrencilerin çalışmalarını daha adil ve objektif bir şekilde değerlendirmeyi mümkün kılar.

MOSS'un Sınırlamaları ve Zorlukları:

Her ne kadar MOSS, kod benzerliği tespitinde güçlü bir araç olsa da, bazı sınırlamaları vardır. Örneğin, MOSS bazen kodun yapısal benzerliklerini yanlış yorumlayabilir veya bazı ince detayları gözden kaçırabilir. Ayrıca, çok karmaşık veya uzun kodlarla çalışırken performans sorunları yaşayabilir. Bu sınırlamalar, MOSS'un kullanımını bazı durumlarda zorlaştırabilir ve yanıltıcı sonuçlara yol açabilir.

Sonuç olarak, MOSS, kaynak kod benzerliği tespitinde önemli bir araçtır ve akademik dünyada geniş kabul görmüştür. Bu aracın sağladığı detaylı analizler ve raporlar, özellikle eğitim alanında, programlama derslerindeki ödevlerin özgünlüğünü değerlendirmede büyük bir yardımcıdır. Bununla birlikte, MOSS'un kullanımı bazı sınırlamalar ve zorluklar içerir, bu nedenle sonuçları yorumlarken dikkatli olmak önemlidir.

2. LİTERATÜR TARAMASI

A Fuzzy-Based Approach to Programming Language Independent Source-Code Plagiarism Detection [2]:

Bu çalışma, programlama dillerinden bağımsız olarak kaynak kod intihali tespitini ele almaktadır. Kaynak kod intihali tespiti, benzer veya aynı kaynak kod parçalarını içeren dosyaların belirlenmesi ile ilgilidir. Bulanık kümeleme yaklaşımları, benzerliğin niteliksel ve semantik unsurlarını yakalama yetenekleri nedeniyle kaynak kod intihali tespitinde uygun bir çözüm sunar.

Araştırmacılar, Fuzzy C-Means ve Adaptive-Neuro Fuzzy Inference System (ANFIS) temelli yeni bir bulanık tabanlı yaklaşım önermişlerdir. Bu yaklaşım, intihali tespit etmek için programlama dili bağımsız bir metod sunar. Önerilen metodun performansı, Self-Organising Map (SOM) ve mevcut en gelişmiş intihal tespit algoritmalarından biri olan Running Karp-Rabin Greedy-String-Tiling (RKR-GST) ile karşılaştırılmıştır. Önerilen yaklaşımın avantajları arasında, programlama dili bağımsız olması ve farklı programlama dillerinde tespit yapmak için herhangi bir ayrıştırıcı veya derleyici geliştirmeye gerek duymaması bulunmaktadır.

Elde edilen sonuçlar, önerilen bulanık tabanlı yaklaşımın, bilinen kaynak kod veri kümelerinde diğer yaklaşımların performansını aştığını ve kaynak kod intihali tespitinde etkili ve güvenilir bir yöntem olarak umut verici sonuçlar sunduğunu göstermektedir.

Semantics-Based Obfuscation-Resilient Binary Code Similarity Comparison with Applications to Software and Algorithm Plagiarism Detection [1]:

Bu araştırma, kaynak veya ikili kod tabanlı mevcut kod benzerliği karşılaştırma yöntemlerinin çoğunun, obfuscation tekniklerine dayanıklı olmadığını belirtmektedir. Programlar arasında benzer veya aynı kod parçalarını tespit etmek, örneğin yasa dışı kod yeniden kullanımını tespit etmek gibi bazı uygulamalarda çok önemlidir. Kod hırsızlığı vakalarında, gelişen obfuscation teknikleri otomatik tespiti giderek zorlaştırmaktadır. Bir başka uygulama, modern zararlı yazılımların tespitini engellemek, ağ iletişimlerini gizlemek ve diğer amaçlarla korumak için yaygın olarak kullandığı kriptografik algoritmaları tanımlamaktır.

Çeşitli kodlama stilleri ve yüksek programlama esnekliği nedeniyle, aynı algoritmanın farklı uygulamaları çok farklı görünebilir, bu da otomatik tespiti çok zorlaştırır, obfuscation kodlarının uygulanması ise bunu daha da güçleştirir. Bu makalede, araştırmacılar, semantik olarak eşdeğer temel blokların en uzun ortak alt dizisine dayalı, yeni bir kavram üzerine kurulu, obfuscation'a dayanıklı bir ikili kod benzerliği karşılaştırma yöntemi önermektedirler. Temel bir bloğun semantiği, bloğun giriş-çıkış ilişkilerini temsil eden bir dizi sembolik formül ile modellenir. Bu şekilde, iki bloğun semantik eşdeğerliği (ve benzerliği) bir teorem çözücüsü tarafından kontrol edilebilir. Daha sonra, temel blokların unsurları olarak en uzun ortak alt diziyi kullanarak iki yolun semantik benzerliğini modellemişlerdir. Bu yenilikçi kombinasyon, kod obfuscation'una karşı güçlü bir dayanıklılık sonucunu doğurmuştur.

Code Similarity Detection using AST and Textual Information [3]:

Bu araştırma, bilgisayar dili kurslarında öğrenciler arasında yaygın olan kod kopyalama sorununu ele almak için kod metni ve AST (Abstract Syntax Tree) tabanlı yeni bir kod benzerliği tespit algoritması önermektedir. Çalışma, öğrencilerin programlama becerilerinin gelişimini engelleyen ve bilgisayar dili kurslarının eğitim kalitesini düşüren bu sorunu çözmeyi hedeflemektedir.

Algoritma, ilk olarak kod metninden yorumları, boş karakterleri ve diğer gereksiz öğeleri kaldırarak "temizleme" işlemi gerçekleştirir ve normalleştirilmiş kod metnini elde eder. Ardından, kelime segmentasyonu, kelime frekansı istatistikleri, ağırlık hesaplaması gibi işlemler yapılır ve Simhash algoritması kullanılarak kod parmak izi elde edilir. Bilgisayar dilinin gramer kurallarına göre, leksikal analiz ve sözdizimsel analiz gerçekleştirilerek AST çıkarılır ve gereksiz bilgiler elenir. Zhang-Shasha algoritması kullanılarak AST düzenleme mesafesi hesaplanır ve AST'ye göre karşılaştırılır. Son olarak, metin benzerliği ve AST benzerliği arasındaki benzerlik derecesi hesaplanır.

Bu yöntemin etkinliğini doğrulamak için, araştırmacılar açık kaynak programlama platformlarından ve LeetCode'dan alınan Python kodları kullanarak, yaygın kod intihal yöntemlerine göre bir test veri seti oluşturmuşlardır. Deneysel sonuçlar, bu yöntemin çeşitli yaygın intihal yöntemlerini tespit etmede yetenekli olduğunu ve ilgisiz kodlar ile intihal edilmemiş kodların deneysel tespitinde düşük benzerlik elde edildiğini göstermektedir. Bu

nedenle, araştırmacılar bu algoritmanın bilgisayar dili kurslarındaki deneysel kod için kod benzerliği tespitinde etkili bir şekilde kullanılabileceğine inanmaktadır.

Bu çalışma, kod benzerliğini tespit etmede AST ve metinsel bilginin kombinasyonunun güçlü bir araç olduğunu göstermektedir. Kodun yalnızca metinsel içeriğine değil, aynı zamanda yapısal özelliklerine de odaklanarak, daha kapsamlı ve derinlemesine bir analiz sunar.

Flowchart-Based Cross-Language Source Code Similarity Detection [4]:

Bu araştırma, kod intihali tespiti ve yazılım fikri mülkiyet korumasında önemli bir yere sahip olan kaynak kod benzerliği tespitine odaklanmaktadır. Bilgisayar programlama öğretiminde öğrenciler, ödevlerini teslim etmek için bir programlama dilinde yazılmış kaynak kodu başka bir dile dönüştürebilirler. Mevcut aynı dilde yazılmış kaynak kodların benzerlik ölçümleri, farklı programlama dilleri arasındaki sözdizimsel farklılıklar nedeniyle çapraz-dil kod benzerliği tespitinde uygulanamaz. Bu arada, mevcut çapraz-dil kaynak kodu benzerliği tespit yaklaşımları, denk kontrol yapısı değiştirme ve gereksiz ifadeler eklemek gibi karmaşık kod obfuscation tekniklerine karşı hassastır.

Bu sorunu çözmek için araştırmacılar, kod akış şemalarına dayalı bir çapraz-dil kod benzerliği tespiti (CLCSD) yaklaşımı önermektedir. Genel olarak, farklı programlama dillerinde yazılmış iki kaynak kod parçası, standartlaştırılmış kod akış şemalarına (SCFC) dönüştürülür ve bunların karşılık gelen SCFC'lerinin ölçülmesiyle benzerlikleri elde edilir. Daha spesifik olarak, ilk olarak, farklı dillerde yazılmış kaynak kodların birleşik akış şeması temsili olarak standartlaştırılmış kod akış şeması (SCFC) modeli tanıtılır. SCFC dilden bağımsızdır ve bu nedenle kaynak kod benzerliği tespiti için ara yapı olarak kullanılabilir. Bu arada, belirli bir programlama dilinde yazılmış kaynak kodun bir SCFC'ye dönüştürülmesi için teknikler sunulmuştur. İkinci olarak, iki SCFC arasındaki benzerliği ölçmek için en kısa yol graf çekirdeği temel alınarak SCFC-SPGK algoritması önerilmiştir. Böylece, farklı programlama dillerindeki iki kaynak kod parçası arasındaki benzerlik, SCFC'ler arasındaki benzerlikle verilir. Deneysel sonuçlar, mevcut yaklaşımlara kıyasla CLCSD'nin çapraz-dil kaynak kodu benzerliği tespitinde daha yüksek doğruluk sağladığını göstermektedir.

Bu çalışma, çapraz-dil kaynak kodu benzerliği tespitinde yeni bir yaklaşım sunmakta ve farklı programlama dillerinde yazılmış kodların yapısal benzerliklerini tespit etmede etkili bir araç olduğunu göstermektedir. Akış şeması tabanlı bu yaklaşım, kodun sadece metinsel içeriğini değil, aynı zamanda yapısal özelliklerini de dikkate alarak, daha kapsamlı ve derinlemesine bir analiz sunar.

3. YÖNTEM

3.1 Abstract Syntax Tree (AST) Analizi

Projede kullanılan AST analizinin temel bileşenlerinden biri, Python kodu parçalarını işleyen *analyze_ast* fonksiyonudur. Bu fonksiyon, kaynak kodun yapısal bir temsilini elde etmek için Abstract Syntax Tree (AST) kullanır. AST, bir programın sözdizimsel yapısını ağaç yapısı olarak temsil eden bir yapıdır ve kodun anlamını daha iyi anlamak için kritik öneme sahiptir.

```python
# İki kod parçasını AST analizi için işleyecek fonksiyon
def analyze_ast(code):
    tree = ast.parse(code)
    nodes = list(ast.walk(tree))
    return nodes
```

Fonksiyonun İşlevselliği:

- **Kodun AST'ye Dönüştürülmesi:**

ast.parse(code): Bu adımda, ast modülünün parse fonksiyonu kullanılarak, verilen Python kodu bir AST'ye (Abstract Syntax Tree) dönüştürülür. parse fonksiyonu, kaynak kodu alır ve kodun yapısal temsili olan bir ağaç nesnesi oluşturur. Bu ağaç, kodun sözdizimsel yapısını temsil eder.

- **AST'deki Düğümlerin Gezinilmesi:**

ast.walk(tree): Elde edilen AST üzerinde gezinme işlemi yapılır. **ast.walk** fonksiyonu, oluşturulan ağaç yapısında, ağacın tüm düğümlerini (nodes) ziyaret eden bir iterator döndürür. Bu işlem, ağacın her bir düğümünün (örneğin, fonksiyon tanımları, ifadeler, sınıflar vb.) detaylı bir şekilde incelenmesini sağlar.

- **Düğümlerin Listesi:**

list(ast.walk(tree)): Burada, **ast.walk** tarafından döndürülen iterator, bir liste haline getirilir. Bu liste, AST'nin tüm düğümlerini içerir ve kodun yapısal analizi için kullanılabilir. Liste formatına dönüştürülmesi, daha sonra bu düğümlerin kolayca işlenmesini ve incelenmesini sağlar.

Fonksiyonun Rolü ve Önemi:

Bu *analyze_ast* fonksiyonu, kaynak kodun yapısal analizinde temel bir rol oynar. AST'nin çıkarılması ve incelenmesi, kodun sadece metinsel içeriğine değil, daha derin yapısal ve mantıksal özelliklerine odaklanılmasını sağlar. Özellikle, kod benzerliği tespiti gibi uygulamalarda, iki farklı kod parçasının yapısal olarak ne kadar benzediğini veya farklılaştığını objektif bir şekilde değerlendirmek için bu analiz önemlidir.

Bu fonksiyonun projedeki kullanımı, kod benzerliği tespit sürecinin temelini oluşturur ve AST üzerinden yapılan analiz, projenin genel metodolojisinde önemli bir yer tutar. Bu fonksiyon sayesinde, projede kullanılan bulanık mantık modeline beslenecek olan AST benzerlik ölçümleri elde edilir ve bu ölçümler, projenin genel kod benzerliği tespit sürecinde önemli bir rol oynar.

3.2Metinsel Benzerlik Hesaplama

Projedeki önemli bir diğer bölüm, metinsel benzerlik hesaplama işlevini gerçekleştiren **calculate_text_similarity** fonksiyonudur. Bu fonksiyon, iki kaynak kod parçası arasındaki metinsel benzerliği değerlendirmek için Python'un **difflib** kütüphanesini kullanır. Metinsel benzerlik, kodun yüzey düzeyindeki benzerliğine, yani kelime ve ifadelerin benzerliğine odaklanır.

```
# İki kod parçası arasındaki metinsel benzerliği hesaplayacak fonksiyon
def calculate_text_similarity(code1, code2):
    sequence = difflib.SequenceMatcher(None, code1, code2)
    return sequence.ratio() * 100  # Yüzde olarak
```

Fonksiyonun İşlevselliği:

- **SequenceMatcher Kullanımı:**

difflib.SequenceMatcher(None, code1, code2): Bu adımda, difflib modülünün SequenceMatcher sınıfı kullanılır. SequenceMatcher, iki string (dizgi) veya liste arasındaki benzerliği hesaplamak için tasarlanmıştır. Bu durumda, **code1** ve **code2** parametreleri olarak verilen iki kaynak kod parçası arasındaki benzerliği tespit etmek için kullanılır. **None** parametresi, SequenceMatcher'ın herhangi bir önceden tanımlanmış veri bloğuna bağlı olmadan çalışmasını sağlar.

- **Benzerlik Oranının Hesaplanması:**

sequence.ratio(): SequenceMatcher sınıfının **ratio** metodu, iki kaynak kod parçası arasındaki benzerlik oranını bir sayısal değer olarak döndürür. Bu değer, 0 ile 1 arasında bir orandır; 1, tam benzerlik anlamına gelirken, 0 hiçbir benzerlik olmadığını gösterir.

- **Sonuçların Yüzde Olarak Dönüştürülmesi:**

return sequence.ratio() * 100: Burada, elde edilen oran yüzde cinsine dönüştürülerek daha anlaşılır bir formatta sunulur.

Fonksiyonun Rolü ve Önemi:

calculate_text_similarity fonksiyonu, iki kaynak kod arasındaki metinsel içeriğin benzerliğini değerlendirir. Bu, özellikle kod kopyalama veya intihal tespiti gibi durumlarda önemlidir. Fonksiyon, kodların yalnızca yapısal özelliklerine değil, aynı zamanda yazılı metinlerinin benzerliğine de odaklanır. Bu, kod benzerliği tespit sürecinde, yapısal analizin yanı sıra metinsel içeriğin de dikkate alınmasını sağlar.

Bu fonksiyon, projenin genel kod benzerliği tespit sürecinin bir parçası olarak, bulanık mantık modeline girdi olarak metinsel benzerlik ölçümü sağlar. Bu sayede, projede elde edilen sonuçlar, hem metinsel hem de yapısal analizlerin bir kombinasyonunu yansıtır, bu da daha kapsamlı ve güvenilir bir benzerlik değerlendirmesi sunar.

3.3 Bulanık Mantık Modeli Oluşturma

Projede yer alan **create_fuzzy_logic_model** fonksiyonu, kaynak kod benzerliğini değerlendirmek için bir bulanık mantık modeli oluşturur. Bu fonksiyon, **skfuzzy** kütüphanesi kullanılarak geliştirilmiş olup, AST benzerliği ve metinsel benzerlik gibi girdileri alarak, genel kod benzerliği hakkında bir çıktı üretir.

```python
# Fuzzy logic modelini oluşturacak fonksiyon
def create_fuzzy_logic_model():
    # Giriş ve çıkış değişkenlerini oluştur
    ast_similarity = ctrl.Antecedent(np.arange(0, 101, 1), 'ast_similarity')
    text_similarity = ctrl.Antecedent(np.arange(0, 101, 1), 'text_similarity')
    overall_similarity = ctrl.Consequent(np.arange(0, 101, 1), 'overall_similarity')

    # Üyelik fonksiyonlarını tanımla
    ast_similarity.automf(3)
    text_similarity.automf(3)
    overall_similarity.automf(3)

    # Kuralları oluştur
    rule1 = ctrl.Rule(ast_similarity['poor'] | text_similarity['poor'], overall_similarity['poor'])
    rule2 = ctrl.Rule(ast_similarity['average'] | text_similarity['average'], overall_similarity['average'])
    rule3 = ctrl.Rule(ast_similarity['good'] & text_similarity['good'], overall_similarity['good'])

    # Kontrol sistemini oluştur ve simüle et
    similarity_control_system = ctrl.ControlSystem([rule1, rule2, rule3])
    similarity_simulation = ctrl.ControlSystemSimulation(similarity_control_system)

    return similarity_simulation
```

Fonksiyonun İşlevselliği:

- **Giriş ve Çıkış Değişkenlerinin Oluşturulması:**

ast_similarity ve text_similarity: Bu değişkenler, AST benzerliği ve metinsel benzerliği temsil eder ve her ikisi de **ctrl.Antecedent** olarak tanımlanır. Bu, her iki değişkenin de modelin giriş değişkenleri olduğunu gösterir.

overall_similarity: Bu değişken, genel kod benzerliğini temsil eder ve **ctrl.Consequent** olarak tanımlanır, yani bu değişken modelin çıktısını temsil eder.

- **Üyelik Fonksiyonlarının Tanımlanması:**

automf(3): Bu metod, her bir giriş ve çıkış değişkeni için üç adet üyelik fonksiyonu otomatik olarak oluşturur. Bunlar genellikle 'düşük' (poor), 'orta' (average) ve 'yüksek' (good) gibi kategorilerdir.

- **Kuralların Oluşturulması:**

Bu adımda, modelin nasıl davranacağını belirleyen kurallar oluşturulur. Örneğin:

rule1: Eğer AST benzerliği 'düşük' veya metinsel benzerlik 'düşük' ise, genel benzerlik 'düşük' olarak değerlendirilir.

rule2: Eğer AST benzerliği 'orta' veya metinsel benzerlik 'orta' ise, genel benzerlik 'orta' olarak değerlendirilir.

rule3: Eğer AST benzerliği 'yüksek' ve metinsel benzerlik 'yüksek' ise, genel benzerlik 'yüksek' olarak değerlendirilir.

- **Kontrol Sisteminin Oluşturulması ve Simüle Edilmesi:**

ctrl.ControlSystem([rule1, rule2, rule3]): Bu, yukarıda tanımlanan kuralları içeren bir bulanık mantık kontrol sistemi oluşturur.

ctrl.ControlSystemSimulation(similarity_control_system): Oluşturulan kontrol sistemi üzerinde simülasyon yapmak için kullanılır. Bu simülasyon, gerçek girdi değerleri verildiğinde modelin nasıl tepki vereceğini test etmeye olanak tanır.

Fonksiyonun Rolü ve Önemi:

create_fuzzy_logic_model fonksiyonu, kod benzerliğini değerlendirmede kullanılan bulanık mantık modelinin temelini oluşturur. Bu model, AST ve metinsel benzerlik gibi farklı girdileri bir araya getirerek, daha geniş ve kapsamlı bir benzerlik değerlendirmesi yapmamızı sağlar. Bulanık mantık yaklaşımı, bu tür karmaşık ve belirsiz verilerle çalışırken, daha esnek ve gerçekçi sonuçlar elde etmeye olanak tanır.

3.4 Kod Benzerliğini Hesaplayan Ana Fonksiyon

Projedeki **compare_codes** fonksiyonu, iki kod parçasının benzerliğini değerlendirmek için temel işlevi üstlenir. Bu fonksiyon, daha önce tanımlanan AST analizi, metinsel benzerlik hesaplama ve bulanık mantık modelini bir arada kullanarak, iki kod parçası arasındaki genel benzerliği hesaplar.

```python
# İki kod parçasının benzerliğini hesaplayacak ana fonksiyon
def compare_codes(code1, code2):
    ast_nodes1 = analyze_ast(code1)
    ast_nodes2 = analyze_ast(code2)

    # AST benzerliğini hesapla
    ast_similarity = 100 - abs(len(ast_nodes1) - len(ast_nodes2)) / max(len(ast_nodes1), len(ast_nodes2)) * 100

    # Metinsel benzerliği hesapla
    text_similarity = calculate_text_similarity(code1, code2)

    # Fuzzy logic modelini oluştur ve girdileri ver
    similarity_simulation = create_fuzzy_logic_model()
    similarity_simulation.input['ast_similarity'] = ast_similarity
    similarity_simulation.input['text_similarity'] = text_similarity

    # Hesaplama yap
    similarity_simulation.compute()

    # Genel benzerlik sonucunu al
    overall_similarity = similarity_simulation.output['overall_similarity']
    return overall_similarity
```

Fonksiyonun İşlevselliği:

- **AST Analizi Yapılması:**

analyze_ast fonksiyonu, her iki kod parçası için çağrılır (**ast_nodes1** ve **ast_nodes2**). Bu işlem, her bir kod parçasının AST'sini (Abstract Syntax Tree) oluşturur ve bu ağaçlardaki düğüm sayısını döndürür.

- **AST Benzerliğinin Hesaplanması:**

AST benzerliği, iki AST'deki düğüm sayısının farkının mutlak değerinin, maksimum düğüm sayısına oranının 100'den çıkarılmasıyla hesaplanır. Bu, yapısal olarak ne kadar benzer veya farklı olduklarını yüzdelik bir oran olarak verir.

- **Metinsel Benzerliği Hesaplama:**

calculate_text_similarity fonksiyonu, iki kod parçasının metinsel içeriğinin benzerliğini hesaplar. Bu, kodun yazılı içeriğinin doğrudan karşılaştırılmasına dayanır ve yüzdelik bir benzerlik skoru üretir.

- **Bulanık Mantık Modelinin Oluşturulması ve Girdilerin Verilmesi:**

create_fuzzy_logic_model fonksiyonu, bulanık mantık modelini oluşturur. Bu model, AST ve metinsel benzerlik skorlarını girdi olarak alır. Bu skorlar, modelin **ast_similarity** ve **text_similarity** girdilerine atanır.

- **Hesaplama Yapılması:**

Modelin **compute** metodu çağrılır, bu, tanımlanan kurallara göre genel benzerlik skorunu hesaplar.

- **Genel Benzerlik Sonucunun Alınması:**

similarity_simulation.output['overall_similarity']: Bu, bulanık mantık modelinin çıktısı olan genel benzerlik skorudur. Bu skor, iki kod parçası arasındaki toplam benzerliği yansıtan yüzdelik bir değerdir.

Fonksiyonun Rolü ve Önemi:

compare_codes fonksiyonu, projenin temel amacını olan "iki kod parçası arasındaki benzerliği değerlendirme" işlemini yerine getirir. Bu fonksiyon, yapısal ve metinsel analizleri bütünleştirerek, kodların genel benzerliğini daha kapsamlı bir şekilde değerlendirir. AST analizi kodun yapısal özelliklerini, metinsel analiz ise doğrudan yazılı içeriğini değerlendirir. Bulanık mantık modeli, bu iki farklı analizin sonuçlarını birleştirerek, daha dengeli ve gerçekçi bir genel benzerlik puanı üretir. Bu yaklaşım, kod benzerliği tespitinde daha kapsamlı ve objektif bir değerlendirme sağlar.

3.5 Genel Benzerlik Durumunu Belirleyen Fonksiyon

Bu bölümde, **classify_similarity** fonksiyonu, hesaplanan benzerlik skoruna dayanarak iki kod parçası arasındaki genel benzerlik durumunu sınıflandırmak için kullanılır. Bu fonksiyon, **compare_codes** tarafından hesaplanan genel benzerlik skorunu alır ve bu skoru belirli aralıklara göre değerlendirir.

```python
# Benzerlik oranına göre genel benzerlik durumunu belirleyen fonksiyon
def classify_similarity(score):
    if score < 40:
        return "Benzer değil"
    elif 40 <= score < 55:
        return "Az benzer"
    elif 55 <= score < 70:
        return "Orta düzeyde benzer"
    elif 70 <= score < 85:
        return "Benzer"
    else:
        return "Çok benzer (Kopya)"

# Benzerlik oranını hesaplayalım ve genel benzerlik durumunu belirleyelim
similarity_score = compare_codes(example_code, github_code)
similarity_classification = classify_similarity(similarity_score)
print(f"İki kod arasındaki genel benzerlik oranı: {similarity_score:.2f}% , {similarity_classification}")
```

Fonksiyonun İşlevselliği:

- **Skor Sınıflandırması:**

Fonksiyon, verilen **score** (benzerlik skoru) parametresine göre bir sınıflandırma yapar. Bu sınıflandırma, belirlenen yüzdelik aralıklara göre kodların birbirine olan benzerlik derecesini belirtir:

- **score** < 40: Skor %40'ın altında ise, iki kod parçası arasında önemli bir benzerlik olmadığı belirtilir ("Benzer değil").
- 40 <= **score** < 55: Skor %40 ile %55 arasında ise, kodlar arasında az miktarda benzerlik olduğu belirtilir ("Az benzer").
- 55 <= **score** < 70: Skor %55 ile %70 arasında ise, kodlar arasında orta düzeyde bir benzerlik olduğu belirtilir ("Orta düzeyde benzer").
- 70 <= **score** < 85: Skor %70 ile %85 arasında ise, kodlar arasında yüksek derecede benzerlik olduğu belirtilir ("Benzer").
- **score** >= 85: Skor %85 ve üzerinde ise, kodların çok benzer veya potansiyel olarak kopyalandığı belirtilir ("Çok benzer (Kopya)").

- **Benzerlik Skorunun Hesaplanması ve Sınıflandırılması:**

compare_codes fonksiyonu, iki kod parçası arasındaki genel benzerlik skorunu hesaplar. Bu skor, **classify_similarity** fonksiyonuna verilir ve skorun hangi kategoriye girdiği belirlenir.

- **Sonuçların Yazdırılması:**

Fonksiyon, hesaplanan genel benzerlik oranını ve bu orana karşılık gelen sınıflandırmayı döndürür. İki kod arasındaki genel benzerlik oranı ve bu oranın ne anlama geldiği ("Benzer değil", "Az benzer", vb.) kullanıcıya bildirilir.

Fonksiyonun Rolü ve Önemi:

classify_similarity fonksiyonu, genel benzerlik skorunu daha anlaşılır ve yorumlanabilir hale getirir.

4. PROJE TESTLERİ

Testin Amacı:

Bu bölüm, projenin "park space detection" konusunda yazılmış kodların benzerliklerini tespit etme kapasitesini değerlendirmeye yöneliktir. Test sürecinde, GitHub'dan elde edilen üç farklı "park space detection" kodu, ChatGPT tarafından oluşturulan bir **example_code** ("park space detection" konusunda) ile karşılaştırılmıştır. Bu karşılaştırmanın amacı, projenin farklı kaynaklardan gelen benzer konulardaki kodlar arasındaki benzerliği ne kadar etkin bir şekilde tespit edebildiğini belirlemektir.

Test Süreci:

- **Kod Örnekleri:**

Üç farklı "park space detection" kodu GitHub'dan temin edilmiştir. Bu kodlar, projenin analiz sürecinde kullanılmak üzere seçilmiştir. Ayrıca, ChatGPT tarafından oluşturulan bir **example_code** bu testin karşılaştırma öğesi olarak kullanılmıştır.

- **Karşılaştırma Kriterleri:**

Her bir GitHub kodu, **example_code** ile karşılaştırılmıştır. Karşılaştırma, AST analizi, metinsel benzerlik hesaplama ve bulanık mantık modelini içermektedir.

- **Testin Yürütülmesi:**

Her bir GitHub kodu için **compare_codes** fonksiyonu kullanılarak, **example_code** ile olan genel benzerlik skoru hesaplanmıştır. Daha sonra, elde edilen bu skorlar **classify_similarity** fonksiyonu ile değerlendirilmiştir.

Test Sonuçları:

- Freedomwebtech github hesabının yolov8parkingspace repository'sinin basic.py dosyasından elde edilen "park space detection" konusunda yazılmış kod ile ChatGPT'ye "park space detection" konusunda yazdırılmış kod arasındaki benzerlik "Benzer değil" sonucunu vermiştir. `***İki kod arasındaki genel benzerlik oranı: 35.77% , Benzer değil***`
- Computervisioneng github hesabının parking-space-counter repository'sinin main.py dosyasından elde edilen "park space detection" konusunda yazılmış kod ile ChatGPT'ye "park space detection" konusunda yazdırılmış kod arasındaki benzerlik "Benzer değil" sonucunu vermiştir. `***İki kod arasındaki genel benzerlik oranı: 35.85% , Benzer değil***`
- Noorkhokhar99 github hesabının car-parking-finder repository'sinin app.py dosyasından elde edilen "park space detection" konusunda yazılmış kod ile ChatGPT'ye "park space detection" konusunda yazdırılmış kod arasındaki benzerlik "Benzer değil" sonucunu vermiştir. `***İki kod arasındaki genel benzerlik oranı: 38.23% , Benzer değil***`

5. SONUÇ

Genel Değerlendirme:

Bu projede gerçekleştirilen testler, "park space detection" konusunda yazılmış kodların benzerlik analizi kapasitesini göstermiştir. GitHub'dan alınan üç farklı kod örneği ile ChatGPT tarafından oluşturulan **example_code** arasında yapılan karşılaştırmalar, projenin farklı kaynaklardan gelen benzer temalı kodlar arasındaki benzerlikleri etkin bir şekilde tespit edebilme yeteneğini kanıtlamıştır.

Test Sonuçlarının Özeti:

Üç test durumunda da ayrı ayrı olmak üzere, GitHub kaynaklı kodlar ile ChatGPT tarafından oluşturulan kod arasındaki genel benzerlik oranları %36.36, %35.85 ve %38.23 olarak hesaplanmıştır. Her durumda elde edilen farklı oranlar, sistemin farklı yazarlar tarafından yazılan kodlar arasındaki yapısal ve metinsel farklılıkları bir şekilde ayırt edebildiğini göstermektedir. Bu sonuçlar, sistemin benzerlik tespitindeki çalışabilirliğini teyit etmektedir.

Uygulamanın Etkinliği ve Kullanım Alanları:

Elde edilen bulgular, özellikle eğitim ve yazılım geliştirme alanlarında, özgünlük ve telif hakkı ihlallerinin tespiti açısından projenin uygulanabilirliğini ve etkinliğini göstermektedir. Sistem, yazılım geliştiricilerine ve akademik kurumlara, öğrenci projelerinin veya yazılım ürünlerinin orijinalliğini değerlendirmede bir araç olarak hizmet edebilir.

Sonuç ve Öneriler:

Projenin test sonuçları, kaynak kod benzerliği analizi konusunda bir başlangıç noktası sunmaktadır. Ancak, böyle sistemler kesin sonuçlar sunmaktan ziyade, bir rehber olarak düşünülmelidir. Sonuçların uzman bir incelemeyle desteklenmesi gerekir. Gelecekteki çalışmalar, farklı programlama dilleri ve daha geniş kod örnekleri üzerinde sistemi test edebilir, algoritmanın doğruluğunu ve kapsamını artırmak için ek parametreler ve analiz yöntemleri entegre edebilir. Bu yaklaşımlar, sistemin genel performansını ve uygulanabilirliğini daha da iyileştirecektir.

6. KAYNAKLAR

[1] L. Luo, J. Ming, D. Wu, P. Liu and S. Zhu, "Semantics-Based Obfuscation-Resilient Binary Code Similarity Comparison with Applications to Software and Algorithm Plagiarism Detection," in IEEE Transactions on Software Engineering, vol. 43, no. 12, pp. 1157-1177, 1 Dec. 2017, doi: 10.1109/TSE.2017.2655046.

[2] G. Acampora and G. Cosma, ‘‘A fuzzy-based approach to programming language independent source-code plagiarism detection,’’ in Proc. IEEE Int. Conf. Fuzzy Syst. (FUZZ-IEEE), Aug. 2015, DOI: 10.1109/FUZZ-IEEE.2015.7337935

[3] Wu Wen, Xiaobo Xue, Ya Li, Peng Gu, and Jianfeng Xu. Code Similarity Detection using AST and Textual Information [J]. Int J Performability Eng, 2019, doi: 10.23940/ijpe.19.10.p14.26832691

[4] Feng Zhang, Guofan Li, Cong Liu, Qian Song, "Flowchart-Based Cross-Language Source Code Similarity Detection", Scientific Programming, vol. 2020, Article ID 8835310, 15 pages, 2020. https://doi.org/10.1155/2020/8835310

[5] Abstract Syntax Tree Library. (2024). Version 3.12 https://docs.python.org/3/library/ast.html

[6] Python Software Foundation. (2024). difflib - Helpers for computing deltas (Python 3.12.1 documentation). https://docs.python.org/3/library/difflib.html

[7] NumPy Developers. (2024). NumPy: A fundamental package for scientific computing with Python (Version 1.26.0) [Software]. https://numpy.org/

[8] scikit-fuzzy Developers. (2024). scikit-fuzzy: Fuzzy Logic Toolbox for Python (Version 0.4.2) [Software]. https://github.com/scikit-fuzzy/scikit-fuzzy

[9] Stanford University. (2024). Measure of Software Similarity (MOSS) [Software]. https://theory.stanford.edu/~aiken/moss/