T.C.

KARADENİZ TEKNİK ÜNİVERSİTESİ

ÖĞRENCİNİN

ADI SOYADI : Tuğba Can

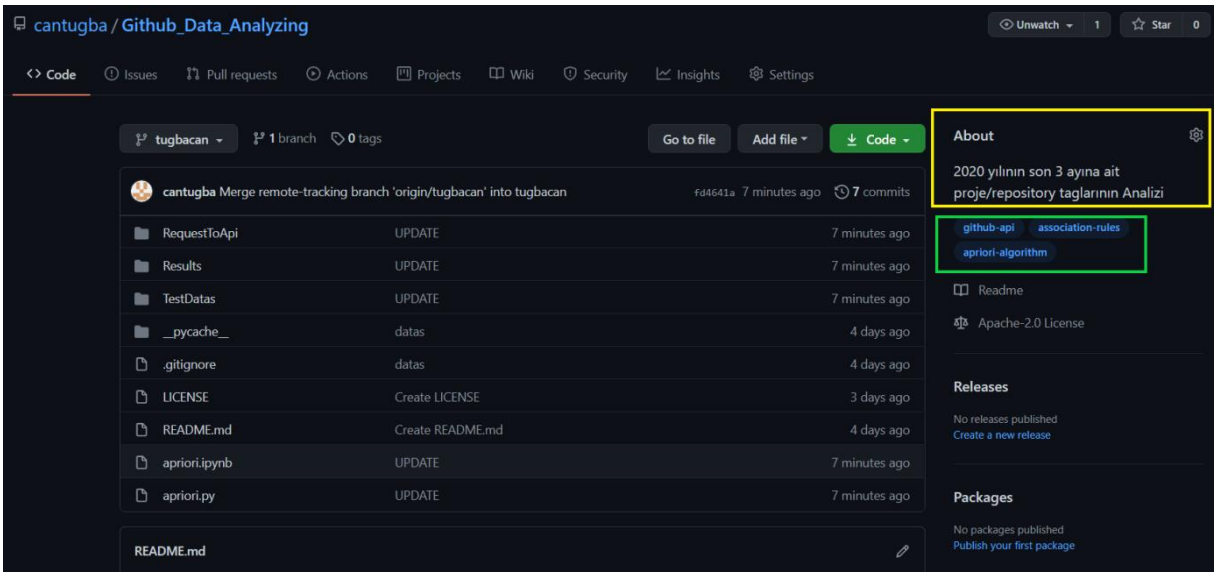
1. Giriş

Bu proje, yazılım geliştiriciler arasında kullanımı oldukça yaygın olan Github üzerindeki projelerin taglarından yola çıkarak 2020 yılının son 3 ayına ait taglar üzerinden analiz yapmayı hedeflemektedir. Veriler Github API ile çekilmiştir.

2. Github

Github, git yazılımı ile entegre olmuş bir depolama alanıdır. Git ise versiyon kontrol sistemidir. Projelerin eski versiyonlarına kolaylıkla geri dönmeye olanak sağlar. Yazılım geliştiriciler arasında kullanımı oldukça popülerdir. Git sistemi, Linux çekirdeğini yazan Linus Torvalds tarafından geliştirilmiştir. Açık kaynak kodlu özgür yazılım ürünüdür. Github ile bir proje üzerinde çalışırken temel bir sürümü oluşturulur ve bu sürüm üzerinden projeye yeni özellikler eklenir. Mevcut proje üzerinde ekip halinde çalışılması ya da proje kodlarının herkes tarafından görülmesine, değerlendirilmesine olanak sağlar.

Github da projeler repository adı verilen depolarda tutulmaktadır. Genel olarak proje depolarının yapısı aşağıdaki gibidir.



Bu depo, örnek projenin yapısını içerir. Sağ tarafta sarı ile gösterilen bölüm projenin açıklamasını içerir. Yeşille özetlenen bölüm proje ile ilgili tagları ya da topicleri içerir.

2.1 Github API

API, geliştiricilerin bulut üzerindeki web araçlarına veya bilgilere erişmek için kullandıkları Uygulama Programı Arayüzleri anlamına gelir. Farklı platformlar arası uygulamaların birbirleriyle iletişimini sağlar.

Github API ise (veya Github REST API), Github ile etkileşim kurmak için kullanılan API'dir. Proje depoları, branch , sorunlar, çekme istekleri ve daha fazlasını oluşturmaya ve yönetmeye olanak tanır. Herkese açık bilgileri (genel proje depoları, kullanıcı profilleri vb.) almak için bu API kullanılabilir. API, Atılan isteğe bağlı olarak verilerin çekilmesini engelleyebilmekte ve bazı istekler için ise kimliği doğrulaması isteyebilmektedir. Github API için temel URL: https://api.github.com/ ‘dur. Atılan istekleri geriye JSON formatında döndürmektedir.

3. VERİ MADENCİLİĞİ

3.1 Genel Tanım

Veri madenciliği bu büyük miktarlardaki verilerin işlenip, analiz edilip bilgiye çevrilmesi olarak tanımlanabilir. Bu gelişmiş analiz sürecinin amacı, bir veri kümesinden bilgi çıkarmak ve çıkardığı bu bilgiyi daha ileri bir kullanım için anlaşılabilir bir yapıya dönüştürmektir. Son yıllarda bilgisayarların ve veri depolama sistemlerinin gelişmesiyle birlikte büyük miktardaki verilere kolaylıkla ulaşılabilmektedir. Büyük miktarlardaki bu verilerin işlenip, analiz edilip bilgiye çevrilebilmektedir

3.2 Veri Madenciliği Ön İşlemler

3.2.1 Veri temizleme; gürültülü ve tutarsız verileri çıkarmak

Veri madenciliği yapılacak ham veri genellikle üzerinde analiz yapılabilecek kadar uygun değildir. Veri kümesi eksik veriler veya analize uygun olmayan tutarsız verileri içerebilmektedir. Bu tutarsız ve hatalı veriler gürültülü olarak değerlendirilmektedir. Böyle durumlarda analizin sağlıklı yapılabilmesi için veri kümesinin söz konusu sorunlardan temizlenmesi gerekecektir. Bu durumda; eksik veri içeren kayıtların silinmesi, değişkenin tüm verileri kullanılarak ortalamasının hesaplanması ve eksik değer yerine bu değerin konması veya eksik değerin tahmin edici algoritmalardan birinin kullanılmasıyla tahmin edilmesi gibi yöntemler kullanılabilmektedir.

3.2.2 Veri bütünleştirme, birçok veri kaynağını birleştirmek

Veri madenciliği süreci büyük veri kümeleri üzerine uygulanmaktadır. Ancak büyük veri kümeleri her zaman direk ulaşılabilir değildir. Bazen veriler dağınık olabilmekte ve bu verilerin birleştirilmesi gerekebilmektedir. Böyle veri tabanlarının birleştirilmesi sürecinde karşılaşılabilecek temel sorun, farklı türdeki verilerin tek türe dönüştürülmesi yani bütünleştirilmesi işlemidir. Aynı türde veriler farklı veri tabanlarında farklı türlerde tutulabilmektedir. Verilerin birleştirilmesi sırasında ise; eğer bütünleştirme yapılmadıysa, aynı veri kümesinde aynı şeyi ifade eden değişkenler sanki farklı değişkenlermiş gibi olabilmekte, bu durum da sağlıklı bir veri madenciliği yapılmasını engellemektedir. Veri madenciliği yapacağımız veri kümesini iyi tanımak bu tür büyük hataların yapılmasını engelleyecektir.

3.2.3 Veri İndirgeme Veri seçme, yapılacak olan analiz ile ilgili olan verileri belirlemek

Veri madenciliğinde bazen analiz işlemleri uzun sürebilir. Sonuçların değişmemesi kaydıyla veri sayısı ya da değişkenlerin sayısı azaltılabilir. Buna veri madenciliğinde veri indirgeme denmektedir. Ancak burada dikkat edilmesi gereken indirgeme işlemi sırasında analizde önemli sayılabilecek değişkenlerin veya verilerin veri tabanından çıkartılmamasıdır. Bu durumda amaçtan sapma söz konusu olacak ve sonuçlar istenildiği kadar sağlıklı olmayacaktır. Veri indirgeme çeşitli biçimlerde yapılabilmektedir. Bunlar; örnekleme, sıkıştırma, boyut indirgeme veya genelleme gibi biçimlerde yapılabilmektedir.

3.2.4 Veri dönüşümü, verinin veri madenciliği teknikleriyle kullanılabilecek hale dönüşümünü gerçekleştirmek

Veri madenciliği sürecinde her zaman veri, değişkenlerin aynı değerleri kullanılarak analize katılmaz. Bir dönüşüm yöntemi kullanarak değişkenlerin normalleştirilmesi veya standartlaştırılması gerekebilir.

3.2.5 Veri madenciliği, seçilen algoritmanın veri kümesi üzerine uygulanması

Veri seti ön işlem süreci ile hazır hale geldikten sonra yapılacak olan, probleme uygun yöntemin ve bu doğrultuda algoritmanın seçilerek veri kümesi üzerinde uygulanmasıdır. Veri dönüşümü sürecinden sonra kullanılacak veri madenciliği modeline karar verilmektedir. Veri kümesi üzerine uygulanan her algoritma farklı sonuç üretmektedir. Burada önemli olan probleme en uygun algoritmanın belirlenebilmesidir. Bu algoritmalar; sınıflandırma, kümeleme ve birliktelik kuralları teknikleri altında toplanmaktadır.

3.2.6 Sonuçları sunma ve değerlendirme

Veri madenciliği algoritmaları veri kümesi üzerine uygulandıktan sonra yapılacak olan işlem, sonuçların değerlendirilmesi ve bu doğrultuda sunulmasıdır. Değerlendirme ve sunum aşamasında sonuçlar grafiklerle desteklenebilir, kullanılan veri madenciliği platformuna göre elde edilen çıktılar sunum için kullanılabilmektedir.

3.3 Veri Madenciliğinde Karşılaşılan Problemler

Veri madenciliğinde büyük boyutlu ve gerçek hayat verileriyle çalışılmaktadır. Bu yüzden VM'de birçok problemle başa çıkmak gerekmektedir. Bu problemler şu şekilde özetlenebilir:

- **Veri tabanı boyutu**:

Veri tabanı boyutları teknolojinin gelişmesiyle büyük bir hızla artmaktadır. Hem özellik sayısı olarak sütun bazında hem de örneklem sayısı olarak satır bazında artmaktadır. Büyük bir veri tabanıyla çalışmak, ondan anlamlı bilgiler elde etmek oldukça zordur. Bununla başa çıkmak için veri kümesinin indirgenmesi veya kullanılan algoritmaların büyük boyutlarla başa çıkabilecek kadar iyi seçilmesi gibi yollara başvurulabilir.

- **Gürültülü veri:**

Veri girişi sırasında oluşan hatalar gürültülü veriye sebep olur. Bu hatalar sonucunda birçok değer yanlış olabilir. Özellikle gerçek hayat verileri düşünüldüğünde bu oldukça kötü bir durumdur. Bundan dolayı gürültülü verilerin saptanması veya etkilerinin en aza indirilmesi önemli bir problemdir.

- **Boş veriler:**

Veri tabanında bulunan boş değerlerdir. Bilinmeyen bir değeri ifade eder. Bu bilinmeyen değerleri çalışmaya dâhil etmek için çeşitli yöntemler kullanılabilir. Boş değerleri dikkate alamamak, boş değerlerin yerine en çok rastlanan değerin yazılması, boş değerlerin yerine ortalama bir değer yazılması, boş değer yerine varsayılan başka bir değer konulması gibi çözümler mevcuttur.

- **Eksik veri:**

Veri tabanında bulunmayan verilerdir. Bu veriler sütun bazlı olabilir. Yani kayıt tutulması gereken önemli bir özellik tutulmamış olabilir. Ayrıca satır bazlı da olabilir. Belirli bir veri kümesinin veri tabanında bulunmaması sonucu oluşan eksik verilerdir.

- **Artık veri:**

Veri kümesindeki gereksiz yere tutulan özellikler artık veri olarak adlandırılır. Bu verilerin elenmesi için özellik seçimi algoritmaları kullanılabilir.

- **Dinamik veri:**

İçeriği sürekli değişen veri tabanları dinamik veri içerir. İçeriğin sürekli değişmesi veri madenciliği uygulamalarında problemlere neden olabilir. Veri tabanı ve veri madenciliği uygulamaları aynı anda çalışır ve bundan dolayı da bazı problemler çıkabilir.

3.4 Veri Madenciliği Modeli : Birliktelik analizi

Veri madenciliğinde sıkça kullanılan yöntemlerden birisidir. Sepet analizi de denir. Alışverişlerde müşterilerin satın alma alışkanlıklarını analiz edip hangi ürünleri birlikte aldıklarını belirler. Bu sayede müşterilerin daha fazla ürün satın alması amaçlanmaktadır. Birlikte olma kurallarını belirli olasılıklarla ortaya koyarak olayların birlikte gerçekleşme durumlarını analiz eden veri madenciliği modeline birliktelik kuralları denmektedir. Birliktelik kuralları yani ilişki analizi, veri kümesindeki bir kaydın diğer kayıtlarla olan bağlantısını açıklayan işlemler dizisidir. Bir kayıt varken bunun yanında başka bir kaydın da var olma olasılığı nedir? Ya da bu iki kayıt varken diğer bir üçüncü veya dördüncü kaydında var olma olasılığı nedir? Şeklindeki sorulara yanıt aramakta ve ortaya çıkardığı bağlantıları birliktelik kuralları olarak tanımlamaktadır. Geçmiş verilerin analiz edilerek bu veriler içindeki birliktelik davranışlarının tespiti ile geleceğe yönelik çalışmalar yapılmasını destekleyen bir yaklaşımdır.

3.4.1 Apiori Algoritması

Apriori algoritması, Agrawal ve Srikant tarafından 1994 yılında geliştirilmiştir. [IBM] Veri Madenciliğinde, birliktelik kuralı çıkarım algoritmaları içerisinde en fazla bilinen ve kullanılan algoritmadır. Algoritmanın ismi, yaygın nesnelerin önsel bilgilerini kullanmasından yani bilgileri bir önceki adımdan almasından "önceki (prior)" kelimesinden gelmektedir. Bu yöntemler, birlikte olma kurallarını belirli olasılıklarla ortaya koyar.

Birliktelik kuralında, öğeler arasındaki birliktelik, destek ve güven kriterleri ile hesaplanır. Destek (Support) kriteri, veride öğeler arasındaki bağıntının ne kadar sık olduğunu belirtir. Destek değeri veri kümesinin eşik değeridir. Veri kümesindeki ;n-elemanlı yaygın öğe kümesi minimum destek kriterini sağlıyorsa bu kümenin bütün alt kümeleri bu kriteri sağlar ve bu ögeyi herhangi bir işleme tabi tutmaya gerek yoktur. Destek kriterinin amacı budur. Güven (confidence) kriteri ise verideki ögelerin hangi sıklıkla bir arada olacağını belirtir. Elde edilen kuralların güvenilirliği, destek ve güven değerleri ile doğru orantılıdır. Algoritmanın adımları aşağıdaki gibidir.

1. Minimum destek sayısı (min.support) ve minimum güven değerinin (min.confidence) belirlenmesi

2. Öğe kümeler içerisindeki her bir öğenin destek değerinin bulunması

3. Minimum destek değerinden daha düşük desteğe sahip olan öğelerin devre dışı bırakılması

4. Elde edilen tekli birliktelikler dikkate alınarak ikili birlikteliklerin oluşturulması

5. Minimum destek değerinden düşük olan öğe kümelerin çıkartılması

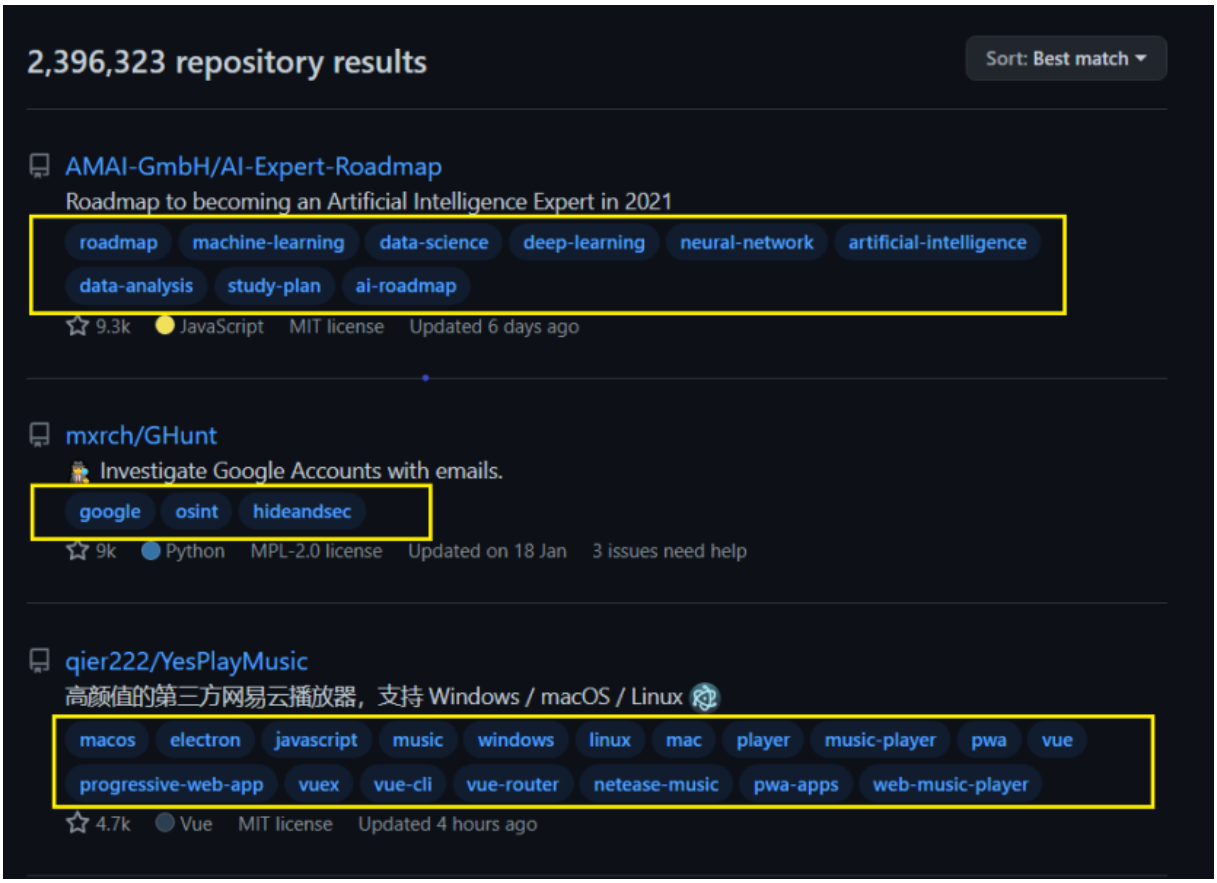
6. Üçlü birlikteliklerin oluşturulması

7. Üçlü birlikteliklerden minimum destek değerini geçenlerin dışındakilerin çıkarılması

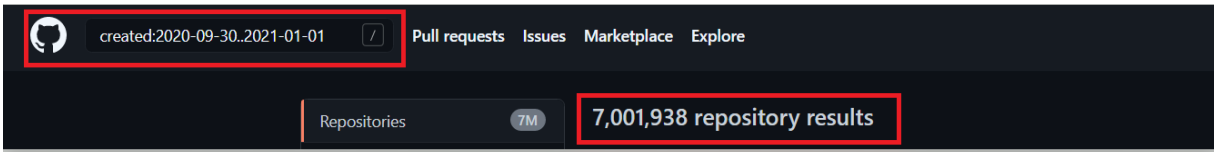
8. Üçlü birlikteliklerden birliktelik kurallarının çıkarılması

4. Yapılan Çalışmalar

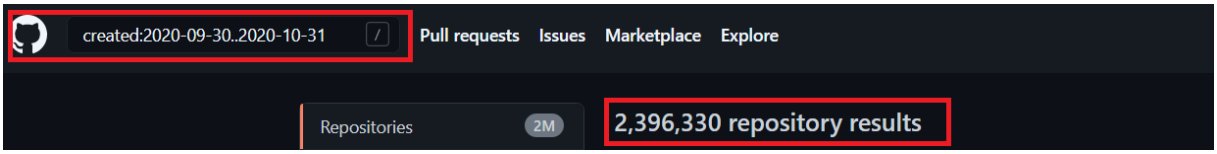
Bu projede 2020 yılının son 3 ayına ait proje taglarının analizinin yapılması hedeflenmiştir. Aşağıdaki şekilde sarı ile belirtilen kısımlar, projeye ait tagları içermektedir.



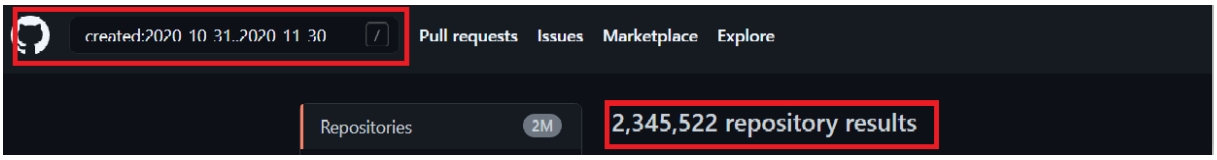
Son 3 ayda toplamda 7,001,938 tane proje oluşturulmuştur. Arama sonucu ve filtreleme formatı aşağıdaki gibidir.



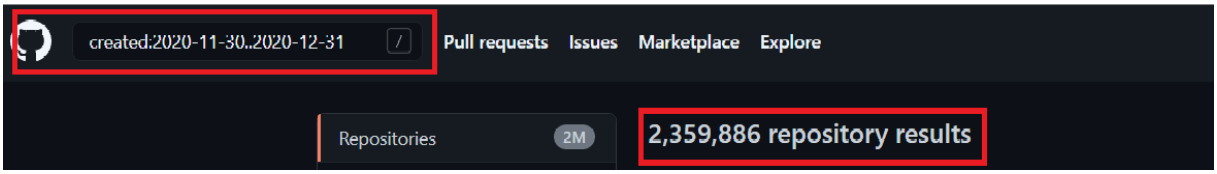
Ancak Github kendi sayfasında bu projelerden sadece 1000 tanesini göstermektedir. Her arama sonucu için maksimum ulaşılabilecek veri sayısı budur. Ekim ayına ait oluşturulan proje sayısı ve arama sonucu filtrelemesi aşağıdaki gibidir.



Kasım ayına ait oluşturulan proje sayısı ve arama sonucu filtrelemesi aşağıdaki gibidir.



Aralık ayına ait oluşturulan proje sayısı ve arama sonucu filtrelemesi aşağıdaki gibidir.



Her sayfada 10 tane proje vardır ve toplamda 100 sayfa vardır. Github'ın sınırlandırmasından dolayı her ay ayrı ayrı filtrelenmiştir böylece 3000'e yakın veriye ulaşılmıştır.

Projelerin taglarına ulaşabilmenin birçok yolu vardır. Bunlardan birisi Github API'ye arama yapılan sayfa üzerinden istek atarak ilgili projelerin Html URL'lerine ulaşıp, bu URL'ler üzerinden projelerin web sayfasına erişerek tagların tutulduğu html taglarını kazıyarak verilere ulaşmaktadır. Web kazıma için sıkça kullanılan BeatifulSoup kütüphanesinden yararlanılmıştır. Bu işlevi gösteren basit bir kod örneği aşağıda verilmiştir.

```
import requests
from bs4 import BeautifulSoup

result =
requests.get("https://api.github.com/search/repositories?q=created%3A2020-09-
30..2020-10-31&page=1&per_page=10")
data = result.json()
for repo in data['items']:

        print(repo['name'])
        print(repo['html_url'])
        r = requests.get(repo['html_url'])
        data = r.text
        soup = BeautifulSoup(data,"html.parser")
        table = soup.find('table',{'class':'topic-tag topic-tag-link '})
        print(table)
        print(repo['language'])
```

Koda ait çıktı aşağıdaki gibidir.

```
In[2]: runfile('C:/Users/tubac/PycharmProjects/GithubData/RequestToApi/WebScraping.py'
AI-Expert-Roadmap
https://github.com/AMAI-GmbH/AI-Expert-Roadmap
None
JavaScript
GHunt
https://github.com/mxrch/GHunt
None
Python
YesPlayMusic
https://github.com/qier222/YesPlayMusic
None
Vue
school-of-sre
https://github.com/linkedin/school-of-sre
None
HTML
MalwareSourceCode
https://github.com/vxunderground/MalwareSourceCode
None
```

Bu işlemin en büyük avantajı API'den bağımsız olarak istenilen taglara veri sınırlaması olmadan erişebilme imkanı sunmasıdır. Dezavantajı ise verileri html tagları arasından kazıma işleminin oldukça yavaş olmasıdır. Bu nedenle başka bir alternatif yol denenmiştir.

Diğer bir yöntem ise Github ın pythona sunduğu PyGithub kütüphanesidir. PyGithub, Github API v3'ü kullanmak içindir. Bu kütüphane ile birlikte, Github kaynakları (depolar, kullanıcı profilleri, organizasyonlar vb.) Python komut dosyalarından yönetebilmektedir. Bu yöntem kullanıcı doğrulamaya ihtiyaç duymaktadır ve sunduğu veriler sınırlıdır. Projeye ait tagların ya da proje deposuna ait verinin belirli kısmını sunmaktadır. Hazır fonksiyonlar nedeniyle koda müdahale etmek mümkün değildir.

Yukarıdaki yöntemlerin yetersizliğinden dolayı ilgili arama sonucu sayfasını Github API formatına dönüştürerek istek atılması yöntemi kullanılmıştır. Bu yöntem de kullanıcı doğrulamasına ihtiyaç duymaktadır. Bu yöntemin en büyük dezavantajı her istek de en fazla 100 tane proje çekmesine izin vermesidir. Avantajı ise diğer yöntemlerden daha hızlıdır ve atılan isteklerin yanıtları JSON formatında olduğu için ilgili projeye ait bilgilerin anahtar değerleri üzerinden hepsine erişmek mümkündür. Özetlemek gerekirse, Github API ile HTTP talebi üzerinden belirli bir projenin açıklamasına, demo bağlantısına, konularına erişebilir ve bu verileri doğrudan kullanabiliriz. Yapılan çalışmadaki istek atılacak API'ye ait URL'in örnek formatı ve kullanıcı doğrulama bilgileri aşağıdaki kodda gösterilmektedir.

```python
import requests

class GithupApi:
```

```
    user = None
    token = None
    number = None
    url = None
    datas = None

    def __init__(self,page_number):
        self.user = "cantugba"
        self.token = "2c74cc548500b6d29de6a74b60fc9bbfd459e3cc"
        self.number = page_number
        self.url = "https://api.github.com/search/repositories?q=created%3A2020-
11-30..2020-12-31&page="+f"{self.number}"+"&per_page=10"
        self.getData()
```

API URL'sindeki per_page değeri her bir istekde çekilecek proje sayısıdır. Maksimum 100 proje çekilebilmektedir ve bu projeleri belirtilen sayfa numaraları ya da arama filtesi sonucundan rastgele çekmektedir. Ancak yapılan işlemlerin karışmaması için bu değer, her sayfadaki proje sayısı kadar belirlenmiştir.

Normalde Github sayfası arama sonucunda bir sayfada en fazla 10 tane proje toplamda 100 sayfalık yani 1000 tane proje verisi sunmaktadır. Bu arama isteğini API üzerinden gerçekleştirdiğimizde ise tek bir istekde en fazla 100 tane proje sunmaktadır. Bu nedenle her 10 sayfada bir yeni bir istek atılmıştır. 1000 tane veriyi çekmek için 5 farklı istek sayfası oluşturulmuştur.

Örnek istek kodları aşağıdaki gibidir.

```
if __name__ == '__main__':
    # Aralık Ayı
    filename1 = 'JsonDatas/DecemberDatas/data.json'
    filename2 = 'JsonDatas/DecemberDatas/data1.json'

    obj_list = list()
    for github_page in range(1, 11):
        githup = GithupApi(page_number=github_page)
        obj_list.append(githup.datas)

    #print(obj_list)
    write_json(obj_list,filename1)

    obj_list2 = list()
    # page_number = 11
    for github_page in range(11, 21):
        githup = GithupApi(page_number=github_page)
        obj_list2.append(githup.datas)

    #print(obj_list2)
    write_json(obj_list2,filename2)
```

API, aynı ip adresi üzerinden atılan istekleri sınırlandırmaktadır yapılan işlemin ardışıklığı ve yoğunluğuna bağlı olarak modemin sıfırlanması gerekebilmektedir. Yapılan denemeler sonucunda her istek sayfası en fazla 2 farklı istek atılabilmektedir. Yani API'ye gönderilen istekler için, bir istek sayfasından en fazla 2 tane veriyi çekecek sınıfın örneği oluşturulabilmektedir. Bu kısıtlamalar arasında veriyi kontrol edebilmek ve çıktıların takibini sağlayabilmek için her istek sayfasının çıktısı 2 farklı JSON dosyasına yazılmıştır.

API üzerinden verileri çekmeyi sağlayan metod aşağıdaki gibidir.

```
def getData(self):
    headers = {"Accept": "application/vnd.github.mercy-preview+json"}
    repos = requests.get(self.url, headers=headers, auth=(self.user,
self.token)).json()
    projects = []
    for repo in repos['items']:
        project = {
            "id": repo["id"],
            "name": repo["name"],
            "url": repo["html_url"],
            "description": repo["description"],
            "topics": repo["topics"]
        }
        projects.append(project)
    self.datas = projects
```

Yukarıdaki kodda verilen ‘topic’ anahtar değeri bu projelere ait tagları içermektedir.

4.1 Veri Madenciliği Aşamaları

4.1.1 Veri Temizleme

Çekilen veriler arasında projelerine tag ya da diğer bir deyişle topic eklememiş kullanıcılar da bulunmaktadır. Analiz işlemi için gereksiz bu tarz gürültülü veriler filtreleme yapılarak kaldırılmıştır. Toplam çekilmiş veri sayısı 2892’dir. Veri setlerine ait proje sayılarının aylara göre dağılımı aşağıdaki gibidir.

Ay	Veri/Proje Sayısı	Gürültülü Veri Sayısı
Ekim	992	453
Kasım	995	505
Aralık	905	424

4.1.2 Veri Bütünleştirme

API üzerinden JSON formatında çekilmiş veriler csv formatına dönüştürülerek birleştirilmiştir. Toplam veri sayısı 1447’dir.

4.1.3 Veri İndirgeme ve Veri Seçme

Veri seti içerisinde projelere ait veriler çekilirken, proje adları diğer bir deyişle depo ve bu projelerin oluşturulduğu aylara ait bilgiler de vardır. Yapılacak olan analiz işlemi projeler içerisinde kullanılan taglar üzerinden olacağı için proje adı ve oluşturuldukları tarihler veri setinden kaldırılmıştır. Aynı zamanda veri setindeki etkisiz veriler yani sonuca bir etkisi olmayacak veriler budanmıştır. Böylece veri setinde sadece analizle ilgili olan veriler kalmıştır.

4.1.4 Veri Dönüşümü

Veriler algoritmanın analiz edebileceği formata dönüştürülmüştür. Yapılan çalışmada CSV dosyasından okunan veriler Pandas kütüphanesine ait çok boyutlu dataframe veri yapısında tutulmaktadır. Bu kütüphane temel olarak zaman etiketli serileri ve sayısal tabloları işlemek için bir veri yapısı oluşturur ve bu şekilde çeşitli işlemler bu veri yapısı üzerinde gerçekleştirilebilir olur.

4.1.5 Seçilen Modelin Veri Kümesi Üzerine Uygulanması

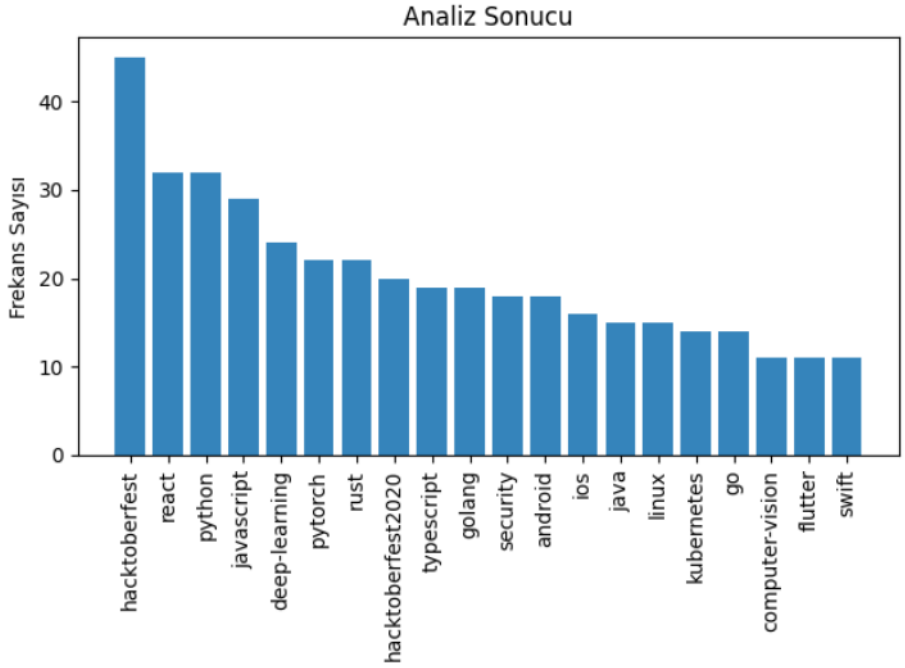
Tagların birlikte kullanılma sıklığının analiz edilmesi için Birliktelik Kuralları algoritmalarından en çok kullanılan Apriori algoritması tercih edilmiştir. Yalnızca en alakalı verileri içeren azaltılmış veriler üzerinde kurallar ve ilişkiler oluşturulmuştur.

İlk aşamada ögelerin frekans sayısı yani tüm sık kullanılan öğe setleri bulunmuştur. En az minimum destek sayısı kadar sık ortaya çıkan öğe grupları da denilebilir. Sonraki aşamada ise sık kullanılan öğe setlerinden yani taglar arasında güçlü ilişkilendirme kuralları oluşturulması hedeflenmiştir. Bu kurallar, minimum desteği ve minimum güveni karşılayan, bu değerlere uyan kurallardır.

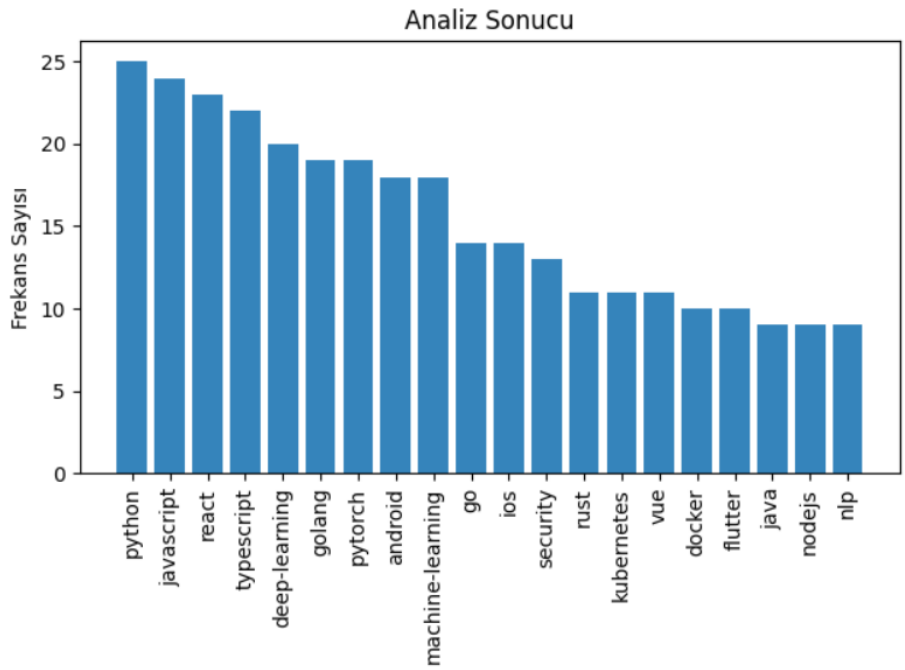
4.1.6 Sonuçları Sunma ve Değerlendirme

Yapılan çalışmada 2020 yılının son 3 ayına ait tagların birlikte kullanılma sıklıkları ve en çok kullanılan tagların aylara göre dağılımının analiz edilmesi hedeflenmiştir. En çok kullanılan tagların analizi frekans sayıları üzerinden yapılmıştır.

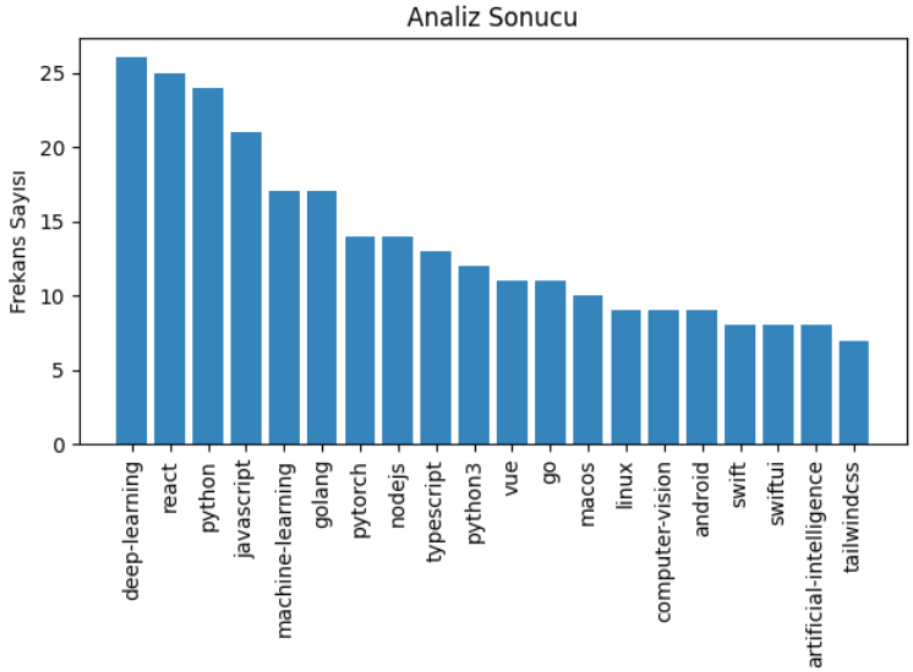
Ekim ayına ait projelerde en çok kullanılan 20 tag aşağıdaki şekilde verilmiştir. “hactoberfest” tagı ekim ayında en çok kullanılan tagdır.



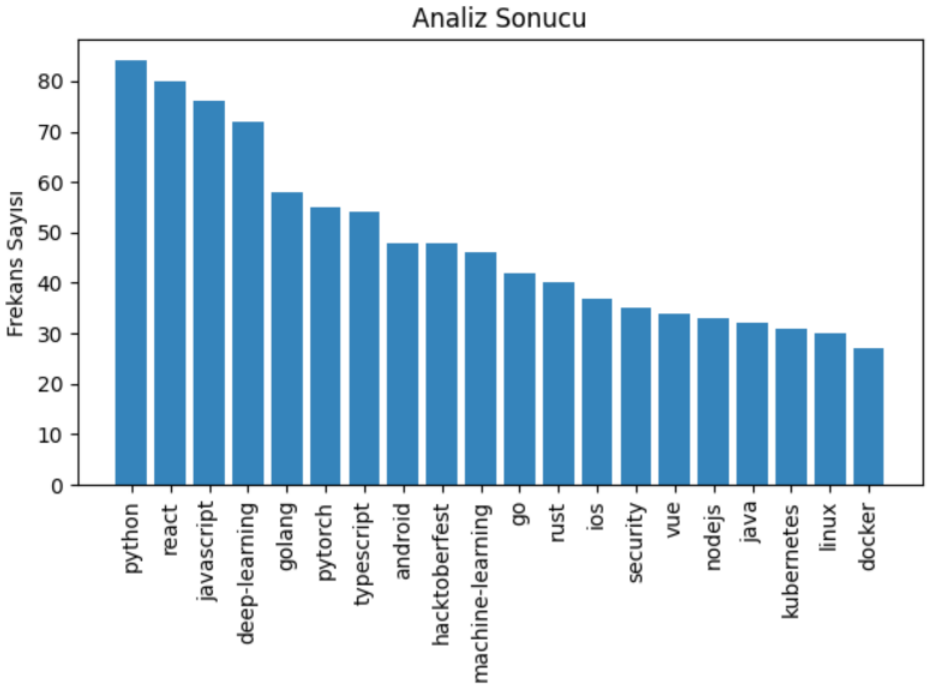
Kasım ayına ait projelerde en çok kullanılan 20 tag aşağıdaki şekilde verilmiştir. En çok kullanılan tag, “python” dır.



Aralık ayına ait projelerde en çok kullanılan 20 tag aşağıdaki şekilde verilmiştir. En çok kullanılan tag, “deeplearning” dir.



Son 3 aya ait projelerde en çok kullanılan 20 tag ise aşağıdaki şekilde gibidir. En çok kullanılan tag, “python” dır.



Hesaplanmış frekans değerleri üzerinden yapılan birliktelik analizi sonucunda Apriori algoritması 98 tane birliktelik kuralı oluşturmuştur. Oluşturulan kurallar ve bu kurallara ait destek ve güven değerleri aşağıdaki tabloda verilmiştir.

Birliktelik Analizi Sonucu	Destek Değeri	Güven Değeri
ai, deep-learning	0.006750241080038573	0.7
ai, machine-learning	0.008678881388621022	0.8999999999999999
android, kotlin	0.011571841851494697	0.30000000000000004
react, angular	0.0057859209257473485	0.6666666666666667
angular, vue	0.0048216007714561235	0.5555555555555556
ios, apple	0.0048216007714561235	0.5555555555555556
attention-mechanism, artificial-intelligence	0.007714561234329798	0.36363636363636365
deep-learning, artificial-intelligence	0.01639344262295082	0.7727272727272727
machine-learning, artificial-intelligence	0.007714561234329798	0.36363636363636365
transformers, artificial-intelligence	0.0048216007714561235	0.41666666666666663
attention-mechanism, attention	0.0048216007714561235	0.625
deep-learning, attention	0.0057859209257473485	0.75
deep-learning, attention-mechanism	0.008678881388621022	0.7499999999999999

awesome-list, awesome	0.008678881388621022	0.5294117647058824
cargo, rust	0.0048216007714561235	1.0
command-line, cli	0.0048216007714561235	1.0
command-line-tool, cli	0.0048216007714561235	0.7142857142857143
cli, golang	0.0057859209257473485	0.3157894736842105
deep-learning, computer-vision	0.010607521697203472	0.5238095238095238
tensorflow, computer-vision	0.006750241080038573	0.33333333333333337
dotnet, csharp	0.0048216007714561235	0.5
javascript, css	0.006750241080038573	0.4375
html5, css3	0.0048216007714561235	0.7142857142857143
ctf-tools, ctf	0.0048216007714561235	0.8333333333333333
flutter, dart	0.010607521697203472	0.7333333333333334
machine-learning, data-science	0.010607521697203472	0.6875
data-science, python	0.007714561234329798	0.5
deep-learning, deep-neural-networks	0.008678881388621022	0.8181818181818181
deep-learning, machine-learning	0.022179363548698167	0.338235294117647
deep-learning, neural-network	0.007714561234329798	0.888888888888889
deep-learning, nlp	0.0057859209257473485	0.375
deep-learning, object-detection	0.0048216007714561235	0.7142857142857143
deep-learning, pytorch	0.024108003857280617	0.3676470588235294
deep-learning, tensorflow	0.01253616200578592	0.6499999999999999
deep-learning, transformer	0.0048216007714561235	0.5
deep-learning, transformers	0.0057859209257473485	0.5
flutter, flutter-package	0.0048216007714561235	1.0
game-engine, game-development	0.0048216007714561235	0.8333333333333333
gamedev, game-development	0.0048216007714561235	0.8333333333333333
react, gatsby	0.0057859209257473485	1.0
go, golang	0.027965284474445518	0.763157894736842
hacktoberfest2020, hacktoberfest	0.017357762777242044	0.4736842105263157
helm, kubernetes	0.0048216007714561235	1.0
javascript, html	0.010607521697203472	0.7857142857142858
html5, javascript	0.006750241080038573	0.6363636363636364
ios, ios14	0.0048216007714561235	0.5555555555555556
ios, macos	0.010607521697203472	0.3333333333333333
ios, scriptable	0.0057859209257473485	0.8571428571428572
ios, swift	0.008678881388621022	0.37499999999999994
ios, widget	0.006750241080038573	0.7
ios14-widget, ios14	0.0048216007714561235	0.5555555555555556
ios14, scriptable	0.0048216007714561235	0.5555555555555556
ios14, widget	0.0057859209257473485	0.6666666666666667
javascript, js	0.0057859209257473485	0.6666666666666667
javascript, nodejs	0.01253616200578592	0.4333333333333333
k8s, kubernetes	0.0048216007714561235	1.0
php, laravel	0.006750241080038573	0.5384615384615384
neovim, lua	0.0048216007714561235	0.7142857142857143
machine-learning, neural-network	0.0057859209257473485	0.6666666666666667

machine-learning, tensorflow	0.007714561234329798	0.4
macos, swift	0.006750241080038573	0.3181818181818182
nodejs, mongodb	0.0048216007714561235	0.45454545454545453
observability, monitoring	0.0048216007714561235	0.35714285714285715
neovim, neovim-plugin	0.006750241080038573	0.6363636363636364
react, nextjs	0.01253616200578592	0.7647058823529411
python3, python	0.006750241080038573	0.35
tensorflow, python	0.0057859209257473485	0.30000000000000004
pytorch, tensorflow	0.006750241080038573	0.35
react, reactjs	0.015429122468659595	0.7272727272727273
react, tailwindcss	0.006750241080038573	0.4375
react, typescript	0.017357762777242044	0.35294117647058826
rust, rust-lang	0.0048216007714561235	1.0
scriptable, widget	0.0057859209257473485	0.8571428571428572
swiftui, swift	0.009643201542912247	0.41666666666666663
tailwindcss, tailwind	0.0057859209257473485	0.6666666666666667
typescript, vue3	0.0048216007714561235	0.38461538461538464
xray, vless	0.0048216007714561235	1.0
vue3, vue	0.006750241080038573	0.5384615384615384
vuejs, vue	0.0057859209257473485	0.6666666666666667
ai, machine-learning, deep-learning	0.0057859209257473485	0.6000000000000001
react, angular, vue	0.0048216007714561235	0.5555555555555556
attention-mechanism, artificial-intelligence, deep-learning	0.0057859209257473485	0.5
deep-learning, artificial-intelligence, machine-learning	0.0057859209257473485	0.35294117647058826
machine-learning, data-science, python	0.0048216007714561235	0.3125
deep-learning, machine-learning, neural-network	0.0057859209257473485	0.6666666666666667
deep-learning, pytorch, machine-learning	0.008678881388621022	0.391304347826087
deep-learning, tensorflow, machine-learning	0.006750241080038573	0.35
deep-learning, pytorch, python	0.006750241080038573	0.7
deep-learning, tensorflow, python	0.0048216007714561235	0.5
deep-learning, pytorch, tensorflow	0.006750241080038573	0.35
ios, scriptable, widget	0.0057859209257473485	0.8571428571428572
java, javascript, python	0.0048216007714561235	0.8333333333333333
machine-learning, tensorflow, python	0.0048216007714561235	0.38461538461538464
machine-learning, pytorch, tensorflow	0.0048216007714561235	0.41666666666666663
react, typescript, nextjs	0.0048216007714561235	0.38461538461538464
react, typescript, reactjs	0.0048216007714561235	0.3125
deep-learning, pytorch, tensorflow, machine-learning	0.0048216007714561235	0.38461538461538464

Minimum güven ve destek değerleri üzerinden bir değerlendirme yapacak olursak veri setindeki en geniş tag kümesi “deep-learning, pytorch, tensorflow, machine-learning” ‘dir.

Yukarıdaki tabloya göre oluşturulabilecek bazı kurallar aşağıdaki gibidir.

- “pytorch” tagını kullananlar, “deep-learning” ,”machine-learning” “python” taglarından en az birini ya da birden fazlasını mutlaka kullanmıştır.
- “artificial-intelligence” tagını kullananlar “deep-learning” tagını mutlaka kullanmıştır. Ancak tam tersi için benzer durum söz konusu değildir.
- “nlp” tagını kullananlar “deep-learning” tagını da kullanmıştır. Tam tersi için benzer durum söz konusu değildir.
- “computer-vision” tagının, “tensorflow” tagı ile birlikte kullanılma sıklığı “deep-learning” ile birlikte kullanılma sıklığından fazladır.
- “kotlin” tagını kullananlar “android” tagını mutlaka kullanmıştır.
- “Typescript” tagını kullananlar mutlaka “react” tagını da kullanmıştır. “reactjs” ya da “nextjs” tagları nı kullanlar “typescript” ve “react” taglarını da kullanmıştır.
- “macos” ya da “swift” tagını kullananlar “ios” tagını da mutlaka kullanmıştır.