Uzman Sistemlere Giris Proje Raporu

Proje Konusu : Steam Platformu Verileri Kullanilarak Oyun Tavsiye Sistemi

Öğrenciler:

Ömer Buğrahan Çalışkan – 17011076

Ömer Aras Kaplan – 17011039

Sistem Gereksinimleri:

Python 3.9

Python dosyalarında gerekli kütüphanelerin import edilmesi

Not: Hocam videoyu grup olarak yan yana çektik.

Video Link: https://youtu.be/y9aDj0mCR_g

Proje Dosyaları Drive Link: https://drive.google.com/file/d/1Lbl9zGwK7WQk2-iQTx3jVmkiClSeJ-2-/view?usp=sharing

Proje Konusu Tanıtımı

Projemizin temel amacı Steam oyun platformu içerisinde bulunan oyunları Steam platformuna kayıtlı olan üyelere önermektir.

Projemizi gerçekleştirirken SteamWebAPI ve SteamSpy api kullanarak (3. Parti) Steam platformunda bulunan toplamda 2000 adet oyun verisi çektik. Ardından örnek user id elde edebilmek için arkadaş sayısı çok olan ve oyun yayını yapan bir kullanıcı seçtik.

Steam'in kendi resmi api servisini kullanarak seçtiğimiz yayıncının arkadaş listesini aldık ve bu arkadaş listesini filtreleyerek 1000 adet kullanıcı id'si elde ettik.

Bu adımın ardından yine resmi steam api sini kullanarak teker teker user id'ler ile istek attık ve kullanıcıların oyun envanterini çekmeyi başardık.

Son olarak önceki adımlarda aldığımız oyun verileri içerisinde bulunan oyun id sini kullanarak SteamSpy api'ye oyunların ortalama oynanma süresi verilerini çektik.

Bu adımın da tamamlanmasıyla veri setimiz tamamlanmış oldu, Verilerimizin özeti aşağıdaki gibidir :

- app_detail : 2000 oyunun ayrıntılı bilgileri
- app_means : Oyunların ortalama oynanma süresi bilgileri
- steam_user_id : 1000 adet oyuncu id
- user_inventory: Seçilen oyuncuların oyun envanterleri

Projemizin geliştirme sürecinde ilk adımda çeşitli api leri kullanarak veri çekme işlemlerimizi tamamladık. Sonrasında kendi algoritmamızı kullanarak her oyuncunun oynadığı oyunlara verebileceği puanı oynama süresi ve ortalama oyun süresi değişkenlerine bağlı olarak hesapladık. Bir sonraki adımda bu puan veri setini kullanarak Collaborative Filtering yöntemiyle her kullanıcı için ona en benzer olan 3 farklı kullanıcı bulduk ve bu 3 en benzer kullanıcı arasında ortalama bir değer hesaplatarak öneri yapacağımız kullanıcıda bulunmayan bir oyunun önerisini yaptık.

Sistemin sayısal başarısını ölçerken kullanıcıların oyunlara puan verdiği veri setinde seçtiğimiz spesifik bir oyuna (CS: GO) en yüksek 100 puan veren kullanıcılardan bu puanları sildik ve bu kullanıcıların indis değerini removed_index şeklinde sakladık. Tekrar hesaplanan öneri veri setindeki oyun bilgilerini removed_index içerisinde aradık ve seçilen oyun CS:GO elde ettiğimizde değişkenimizi arttırdık. Test sürecinde silinen oyunculara CS:GO önermesini bekliyorduk, aldığımız veriler doğrultusunda oyunculara %84.81 oranında doğru sonuç verdiğimizi hesapladık bu da sistemimizin sayısal başarısı olmuş oldu. Test aşamasında 78 adet kullanıcı test edilmiştir.

Projemizi gerçekleştirirken veri toplama aşaması için internet üzerinden python kullanılarak nasıl veri çekebileceğimizi araştırdık, bu konuda github üzerinde bulunan benzer projeleri inceledik ve @annelovepeace kullanıcısının benzer örnek için kullandığı veri toplama yönteminden yararlandık, aynı zamanda program içerisinde uzun süren işlemlerin tamamlanma yüzdesini öğrenebilmek için show_work_status adlı fonksiyonunu kullandık.

Kullanıcıların oyunlara verebileceği puanı hesaplarken ortalama oyun süresi ve kullanıcının oynadığı süreleri baz alarak kendi algoritmamızı geliştirdik.

Benzerlik aşamasında ise internette bulunan user-based collaborative filtering öneri sistemi örneklerini inceledik.

Kullanıcı Puanı Oluşturma Algoritması:

Steam kullanıcılara oyun ile ilgili geri dönüş alabilmek için sadece beğenip beğenilmediğini ya da konuyla ilgili yorumlarını bırakabileceği bir alan sunar fakat Steam kullanıcılara oyunlara puan verebilmesi için herhangi bir sistem sunmaz. Bu durumdan dolayı kullanıcılardan puan elde edebilmek için kendi algoritmamızı oluşturduk. Algoritmamız kullanıcının oyuna harcadığı sürenin oyunun ortalama oynanma süresine bölünmesiyle hesaplanmaktadır. Bu denklemden çıkan sonuçlar belli iki değer arasında olmadığı için bu verilere 0-100 arasında normalizasyon uyguladık. Bu hesaplama sırasında oyuncunun bütün oyunlarına harcadığı toplam süreyi denklem dışında bıraktık çünkü normalizasyon yapılacak dizideki elemanları sürekli sabit sayıyla bölersek normalize edildiğinde sonuç değişmeyecekti.

User Based Collaborative Filtering:

User Based Collaborative Filtering yöntemi, seçilen kullanıcının diğer kullanıcılar ile ortak sahip oldukları nesnelere verdikleri puanlara bakarak en benzer kullanıcıların bulunması ve seçilen kullanıcının envanterinde bulunmayan bir nesneye en benzer kullanıcılarının verdiği puanları benzerlik oranına göre hesaplatarak verebileceği puanın tahmin edilmesi esasına dayanır ve sonucunda en yüksek hesaplanan puana sahip olan nesne kullanıcıya önerilir. Bu öneri esnasında oyun önerilecek kullanıcının envanterinde bulunmayan oyunlar esas alınır.

Program Çalıştırılma Aşamaları:

Öncelikle Veri setinin oluşturulabilmesi için sırasıyla:

- app_detail_crawl.ipynb
- user_inventory_crawl.py
- getAppMeans.py

Dosyaları çalıştırılır ve gerekli veriler data klasörü içerisinde oluşmuş olur.

Oyun detayları steam resmi apisine istek atarak getirilir

```
with open(path_app_detail, 'w') as f:
    for app_id in lst_app_id:
        url_app_detail = ('http://store.steampowered.com/api/appdetails?appids=%s') % (app_id)
        for i in range(3):
            try:
                r = requests.get(url_app_detail)
                result = r.json()
                break
            except:
                time.sleep(5)
                pass
        f.write(json.dumps(result))
        f.write('\n')
        show_work_status(1, total_count, current_count)
        current_count += 1
```

Kullanıcı oyun envanteri steam hesabı üzerinden alınan api key ve örnek kullanıcı id lerinin bulunduğu bir dosya kullanılarak getirilir.

```python
def worker(user_id):
    dic_temp = {}
    base_url = 'http://api.steampowered.com/IPlayerService/GetOwnedGames/v0001/'
    params = {
        'key' : 'C039C46919B6152200FF6FCD42241897', # Steam web API key
        'steamid' : user_id.strip(),
        'format' : 'json' }
    r = requests.get(base_url, params = params)
    user_inventory = r.json().get('response').get('games')
    dic_temp.update({user_id.strip():user_inventory})
    time.sleep(1)
    return dic_temp
```

Veri toplamanın son aşamasında oyunların ortalama oynanma süresini bulabilmek için oyun id leri kullanılarak Steamspy api ye istek atılır ve verileri dosyaya yazılır.

```python
for x in temp:
    ploads = {'request': 'appdetails', 'appid': temp[i][0]}
    r = requests.get('http://steamspy.com/api.php', params=ploads)

    temp_dict = {
        'appid': temp[i][0],
        'name': temp[i][1],
        'median': r.json().get('median_forever')
    }
    i += 1
    show_work_status(1, len(temp), i)
    json.dump(temp_dict, app_means)
    app_means.write("\n")
    time.sleep(1)
app_means.close()
```

Her oyuncunun oynadığı oyunlara verdiği puanı hesaplamak için yukarıda açıklamış olduğumuz algoritma ile değerler hesaplanır bu hesaplamanın yapılması için getPlayerRatings.py dosyası çalıştırılmalıdır. Sonuç dosyası player_ratings.csv dosyasında oluşmaktadır.

```
for player_line in player_details_file:
    show_work_status(1, 1000, i)
    i += 1
    player_common_games = []
    player_game_ratings = []
    player_games = json.loads(player_line.strip())
    player_ids.append(list(player_games)[0])
    player_games = list(player_games.items())[0][1]
    means_file = open("data/app_means.txt")
    for mean_line in means_file:  # oyunlari appmeansten al
        flag = False
        mean_game_props = json.loads(mean_line.strip())
        mean_line_appid = list(mean_game_props.items())[0][1]  # Oyunun appid
        mean_line_median_amount = list(mean_game_props.items())[2][1]  # Oyunun median degeri
        if player_games is None:
            continue
        for game in player_games:  # oyuncunun oyunlarini al
            if game['appid'] == int(mean_line_appid):
                if mean_line_median_amount == 0:
                    player_game_ratings.append(None)
                else:
                    player_game_ratings.append(game['playtime_forever'] / mean_line_median_amount)
                    player_common_games.append(game['appid'])
                    flag = True
                    break
        if flag == False:
            player_game_ratings.append(None)
            player_common_games.append(None)
    means_file.close()
    arrayOfRatings.append(normalize(player_game_ratings,1,100))
    all_player_ratings.write(str(normalize(player_game_ratings,1,100)))
    all_player_ratings.write("\n")
```

Collaborative Filtering uygulanma ve önerinin yapılabilmesi için önceki adımda oluşturulan player_ratings.csv dosyası gerekmektedir. Bu dosya kullanılarak kullanıcılara benzer kullanıcılar hesaplanır ve en çok önerilen 5 adet oyun recommendation.csv dosyasına yazılır. Bu işlemler recommendation.py dosyasında yapılmaktadır.

```
##########
#Collaborative Filtering---User Score Based
##########

def similar_users(user_id, matrix, k=3):...


def recommend_item(user_index, similar_user_indices, matrix, items=5):...


total_count = len(all_users_list)
show_work_status(0, total_count, 0)
recommended = []
for user_index in range(0, total_count):
    similar_user_indices = similar_users(user_index, df)
    recommended.append(recommend_item(user_index, similar_user_indices, df).values.tolist())
    show_work_status(1, total_count, user_index)
```

test.py Dosyası:

Sistemin sayısal başarı testi için oluşturulmuş bir dosyadır.

player_ratings.csv okunduktan sonra CS:Go oyununa 100 tam puan veren kullanıcıların bu 100 puanı 0'a çekilir ve bu veri setine göre bir öneri yapılır. Silinen kullanıcıların indexleri tutulduğundan dolayı yeni öneri verilerinde önerilen oyunların silinen oyun olup olmadığına bakılır ve kaç tanesine öneri yapıldığını

bir döngü içerisinde hesaplayarak yüzde formatında bir değer sunar. Bizim durumumuzda başarı oranı %84.81 olarak hesaplanmıştır.

```
removed_indices = []
for column in range(0,len(df)):
    if df.iloc[column][1] == 100.0:
        df.iloc[column][1] = 0.0
        removed_indices.append(column)
print(removed_indices)
```

```
success_rate = 0

for index in removed_indices:
    game_name = recommended_df.iloc[index][0][1]
    if game_name == 'Counter-Strike: Global Offensive':
        success_rate+=1

success_rate = (success_rate/len(removed_indices)) * 100

print(success_rate)
```

```
[10, 32, 80, 90, 109, 143, 210, 220, 229, 232, 238, 241, 245, 255, 257, 261, 276, 278, 286, 289, 336, 341, 342, 356,
[>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] 100.00%

84.81012658227847
```

Projeye Dair Yorumlar

Öncelikle projede kendi skor algoritmamızı oluşturmak oldukça zevkliydi. Ek olarak projeyi yaparken karşımıza çıkan zorluklar bizi geliştirdi. Json dosyalarıyla çalışmayı, pandas kütüphanesini, data frameler ile çalışmayı ve sklearn kütüphanesiyle çalışmayı öğrendik. Son olarak yaptığımız projeyi ileride kendimiz için de kullanabileceğimiz için bu projeyi yaparken oldukça heyecanlıydık.