T.C.

MUĞLA SITKI KOÇMAN ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

YAPAY ZEKA ANABİLİM DALI

ZEKİ OPTİMİZASTON YÖNTEMLERİ FİNAL ÖDEVİ

KONU: Diferansiyel Evrim ile Future Selection İşlemi Yapılması

ÖĞRENCİ ADI-SOYADI: Okan İhsan Bağrıaçık

ÖĞRENCİ NUMARASI: 214225005

Giriş

Analitik dünyasında kullandığımız uygulamalar ve modeller birer araçtır. Hepsi bizim elimizdeki problemimizi çözmeye yarayacak araç ve gereçlerdir. En optimum sonuca ulaşırken ihtiyacın olan uygulama ve çözümleri kullanabilirsin. Analitik modeller yani algoritmalar her yerde aynı şekilde çalışır. Benim bilgisayarımda farklı senin bilgisayarında farklı çalışmaz.
Analitik araçların kullanımı açısından yorumlayacak olursak algoritmalar farklı ortamlarda yine aynı şekilde çalışır. R'daki sinir ağı modeli ile Python'daki sinir ağları modelleri aynı parametrelerle aynı işi yapar. Analitik araçların birbirine karşı farklı üstünlükleri vardır ama bu aynı koşullarda model başarı performansı değildir.

Elimizdeki problemin çözümü için günümüzde sürekli daha iyi sonuç veren algoritmalar çıkıyor ve veri bilimi dünyası geliştikçe de çıkacaktır. Dünyanın iki farklı ucundaki kişiler bu yeni algoritmaları ve paylaşım ekosistemindeki tüm algoritmaları problemini çözmek için kullanabilirler. Bu kısımda aynı veri seti ve aynı algoritmalar olduğu için çözümde bir farklılaşma yoktur.

Aynı problem için sonuçlarda farkı oluşturan o model için kullandığın özelliklerdir. Öznitelikler diye de söylebiliriz. Yani model için kullanacağın girdi değişkenleridir. Hem özellik oluşturma kısmı hem de o özelliklerden hangilerini modelde kullanacağın kısım, senin aynı problemde farklı sonuçlar elde etmeni sağlar.
İşte bu yüzden en baştan doğru değişkenler oluşturmak ve seçmek bu işin en önemli kısmıdır. Öyle iyi ve kaliteli özellikler ortaya çıkarır ve hazırlarsın ki model yerine kural bazlı kuracağın yapılar bile çok iyi sonuç verecektir. Başarının ana kriteri doğru özellikleri bulup temizlenmiş şekilde kullanarak modeli kurmaktır.

NEDEN İHTİYACIMIZ VAR?

Neden değişkenleri seçmeye ihtiyacımız var ?

• Çok fazla değişken kullanmak **modelin performansı**nı düşürebilir. En iyi performans sonucu elde etmek için, en optimum değişkenleri belirleyip onlar ile model kurmak gerekir. Çok fazla değişken kullanmak her zaman iyi birşey değildir.

• Daha **kolay anlaşılır model** elde etmek için ihtiyacımız var. Modellerin çok karmaşık olması çok özenilir bir durum değildir, ne kadar basit ve açıklayıcı olursa o kadar iyidir diyebiliriz. Model sonuçlarını anlayarak ve ona göre de aksiyonlar belirlenip o modelden fayda sağlanmasıdır zaten amacımız. Model sonuçlarını gerçek hayata uygulamak daha kolay olacaktır.

• Daha **hızlı çalışan model** elde etmek için ihtiyacımız var. Örnek veri setlerinde çalışmanın dışında gerçek hayata geldiğimizde, sürekli çalışan ve sonuç üreten modellere ihtiyacımız olabilir ve o zaman modelin hız performansı çok önemli hale gelir. Özellik azaltarak model başarı performansından ödün vermeden daha hızlı çalışan modeller elde edebiliriz. Hatta bazen model başarı performansından az ölçüde feragat edip modelin hız performasını anlamlı ölçüde arttıracak sonuçları tercih edeceğimiz yerler olabilir.

• **Aşırı öğrenme riski**ni azaltır. Çok fazla değişken ile model karmaşıklığı artarak ezberlemeye geçebilir. Daha az değişken kullanmak aşırı öğrenme riskini de azaltmış olur.

Öznitelik Seçimi (Feature Selection) Teknikleri

Öznitelik bir veri seti içerisinde bulunan ve hedeflenen model çıktısını oluşturmamızı sağlayacak olan her bir kolon/sütundur.

Öznitelik;

- Bir banka müşterisinin churn tahmini için alınan veri setindeki müşterinin üyelik senesi, vade hesap sayısı, yaş , cinsiyet, meslek vb.
- Alışveriş sitesinde kullanıcılara ürün öneri modeli için ise gezilen ürün kategorileri, satın alma sıklığı vb. kolonlar olabilir.

Öznitelik seçimi , veri seti içerisinde en yararlı öznitelikleri seçme ve bulma sürecidir. Bu işlem Makine Öğrenmesi modelinin performansını çok fazla etkilemektedir.

Gereksiz öznitelikler;

- Modelinizin Training süresini arttırabilmektedir.
- Modelimizin basit ve açıklanabilir olmasını isteriz. Fazla sayıdaki öznitelik modelinizin yorumlanabilirliğini azaltabilmektedir.
- Model başarısının training veri setinde overfitting nedeniyle yüksek ancak test veri setinde ise düşük olmasına sebep olabilmektedir. Test veri setinde gelecek olan kayıtlar training veri setindeki kayıtlar ile benzerlik göstermediği durumda modelde hata oranı yüksek olacaktır.

Model tasarımınızda en önemli ve ilk adım mutlaka veri temizleme ve öznitelik seçimi olmalıdır.

Öznitelik seçim metotlarının büyük bir kısmı 3 ana kategoriye ayrılabilir.

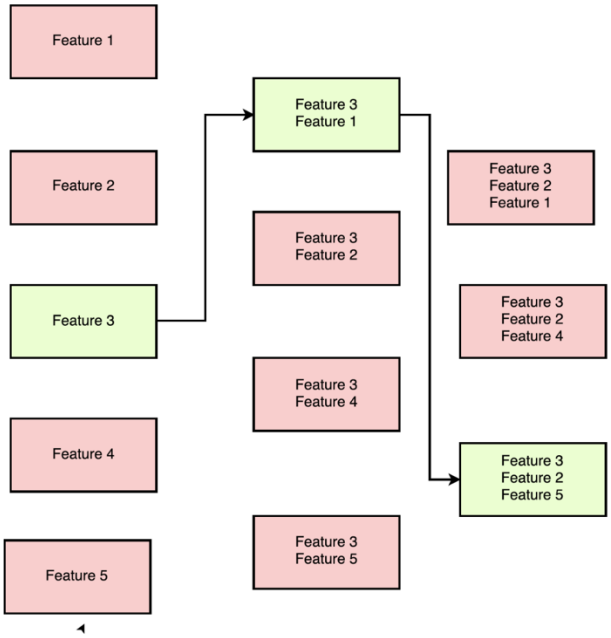
- ***Filtre Yöntemleri :*** Özniteliklerin önemini hesaplamak için öznitelik ile hedef değişken arasındaki ilişkiyi dikkate alan yöntemlerdir. Yapılan işlemler sonucunda veri setimizde filtreleme yaparız ve ilgili özellikleri seçerek bir alt küme oluştururuz. En sık kullanılan yöntemler ise Pearson Correlation ve Ki-Kare Yöntemidir.

1. Pearson Correlation : Doğrusal olarak ilişkili değişkenler arasındaki ilişkinin derecesini ölçmek için en yaygın kullanılan korelasyon ölçütüdür.

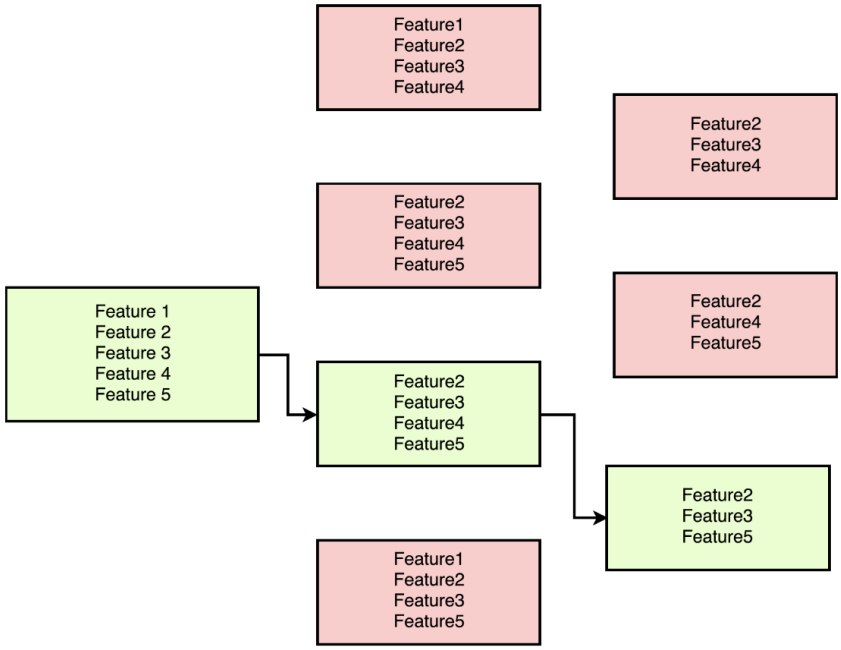
2. Ki-Kare Testi Yöntemi : Bu test sadece kategorik değişkenler için çalışmaktadır. Örneğin: Cinsiyet (Kadın — Erkek), Renk (Siyah, Beyaz, Mavi) . Sayısal değerler için çalışmamaktadır.(Boy,kilo vb) Bu istatistiksel yöntem ile hedef değişken ve öznitelik arasında ilişki olup olmadığını test sonucunda elde ettiğimiz p değeri ile belirleyebiliriz. Eğer p value <0.05 ise bu öznitelik hedef değişkenimiz için bağımlı bir değişkendir, eğer p≥0,05 ise bağımsız bir değişkendir.

- ***Sarmalayıcı Yöntemler(Wrapper — Based)*** bir alt özellik grubuna sahip modeller oluşturur ve model performanslarını ölçer. Bunun için iki tip arama stratejisi vardır ;

- Forward/İleri Arama: Boş bir öznitelik kümesi ile başlayarak her seferinde gruptaki öznitelikleri tek tek ekleyerek özniteliklerin kalitesi test edilir.



- n öznitelikli bir veri setinde;
- İlk adımda en iyi tahminleme yapan tek bir öznitelik seçilir.
- İkinci adımda ilk seçilen öznitelik ile beraber en iyi tahminleme yapan 2. öznitelik belirlenir . Böylece en iyi tahminleme yapan 2'li alt grup oluşturulmuş olur.
- Bu işlemler en iyi tahminleme yapan “m” adet öznitelik kombinasyonu bulunana kadar devam eder.
- Backward/Geri Arama(Recursive Feature Elimination) : Adından da anlaşılacağı gibi, bu yöntem , en iyi öznitelik alt kümesi bulunana kadar tüm öznitelik kümesinden başlayarak adım adım en kötü performans gösteren öznitelikler elenir.



“n” adet öznitelikli bir veri setinde;

- Tüm veri setindeki öznitelikler alınır ve en az performans gösteren öznitelik kaldırılır.
- İkinci adımda ilk adımda belirlenen alt gruptaki yine en az performans gösteren öznitelik kaldırılır.
- Bu işlemler en iyi tahminleme yapan “m” adet öznitelik kombinasyonu bulunana kadar devam eder.
- ***Embedded Metotlar:*** Öznitelik seçimi, bazı Makine Öğrenimi modellerinin sağladığı iç görülerle de elde edilebilir. Örneğin, Lasso ve RF algoritmalarının kendilerine ait öznitelik seçim metotları bulunmaktadır.

- **LASSO Lineer Regresyon ,** Lineer Regresyonun maliyet fonksiyonuna fazladan bir terim eklenerek gerçekleştirilir. Bu, overfitting olmayı önlemenin yanı sıra, daha az önemli özelliklerin katsayılarını sıfıra indirir.

$$\min_{\theta} \frac{1}{2} \sum_{i} (y_i - X_i\theta)^2 + \lambda ||\theta||_1$$

İmplamantasyon

İmplamantasyon kısmı Colab üzerinden yapılmıştır ve her aşamada yazıla kodların ve yapılan işlemlerin açıklamaları metin kutuları içerisinde belirtilmiştir. Bu notebook üzerinden alınan ekran görüntüleri rapora eklenmiştir.

İlk olarak Python dili ile yazılmasına karar verilmiş olan Future Selection için gerekli kütüphanelerin implamantasyonu yapılmıştır.

```
import numpy as np
import pandas as pd
from sys import exit
import matplotlib.pyplot as plt
```

Daha sonrasında veri seti içerisinde her seferinde aynı verilerin alınması ve incelenmesini engellemek amacıyla rastgele seçim methodları yazılmıştır.

```
rand = np.random.rand
choice = np.random.choice
clip = np.clip
argmin = np.argmin
min = np.min
```

Veri seti içerisinden verilerin alınması ve daha sonrasında Diferansiyel Evrim Algoritması tarafında kullanılması için bir definasyon yazılmıştır. Bu definasyon verileri alarak ebeveynleri oluşturmaktadır.

```
# veri olusturucu fonksiyon
def veri_olustur(x):
        return x[0]**2.0 + x[1]**2.0
```

Daha sonrasında Diferansiyel Evrim Algoritmasında kullanılacak olan mutasyon,sınır kontorlü ve çaprazlama operasyonları için ayrı ayrı 3 definasyon oluşturulmuştur.

```
# mutasyon fonksiyonu
def mutasyon(x, F):
        return x[0] + F * (x[1] - x[2])

# boundary kontrol eden fonksiyon
def bounds_kontrol(mutated, bounds):
        mutated_bound = [clip(mutated[i], bounds[i, 0], bounds[i, 1]) for i in range(len(bounds))]
        return mutated_bound

# caprazlama yapan fonksiyon
def caprazlama(mutated, target, dims, cr):
        # her dim icin ayri random value uret
        p = rand(dims)
        # binom caprazlama
        trial = [mutated[i] if p[i] < cr else target[i] for i in range(dims)]
        return trial
```

Diferansiyel Evrim algoritması içinde bir definasyon oluşturulmuştur. Burada öncelikle bir random seçim uygulanmaktadır. Amaç veriseti içerisinden verilerin seçilmesi ebeveyn olarak kullanılmasıdır. Bu ebeveynler başlangıç popülasyonu içersinde yer alır ve ilk iterasyon bu Futureler üzerinden yapılır. Kullanıcı tarafından belirlenen iterasyon sayısı kadar iterasyon işlemi başlar. En iyi 3 aday seçilir ve Diferansiyel Evrim algoritması operasyonları uygulanır. Bu işlem uygulandıkça döngü içerisinde ki en iyi değerler seçilir ve saklanır. Bu değerler veri seti içerisinde ki kullanılabilir verileri temsil etmektedir.

```
def diferansiyel_evrim(pop_size, bounds, iter, F, cr):
        # random secim yap
        pop = bounds[:, 0] + (rand(pop_size, len(bounds)) * (bounds[:, 1] - bounds[:, 0]))
        # baslangic popunu ayarla
        obj_all = [veri_olustur(ind) for ind in pop]
        # baslangic popundaki en iyiyi bul
        best_vector = pop[argmin(obj_all)]
        best_obj = min(obj_all)
        prev_obj = best_obj
        # iter sayisi kadar
        for i in range(iter):
                # pop sayisi kadar iterasyonlari calistir
                for j in range(pop_size):
                        # 3 aday bul
                        candidates = [candidate for candidate in range(pop_size) if candidate != j]
                        a, b, c = pop[choice(candidates, 3, replace=False)]
                        # mutasyon yap
                        mutated = mutasyon([a, b, c], F)
                        # mutasyondan sonraki lower ve upperleri bul
                        mutated = bounds_kontrol(mutated, bounds)
                        # caprazlama yap
                        trial = caprazlama(mutated, pop[j], len(bounds), cr)
                        # herefi belirle
                        obj_target = veri_olustur(pop[j])
                        # hedef degerini hesapla
                        obj_trial = veri_olustur(trial)
                        # hedeften secim yap
                        if obj_trial < obj_target:
                                # hedefi deneme vektoruyle degistirin
                                pop[j] = trial
                                # yeni degere ata
                                obj_all[j] = obj_trial
                # her iter sonrasi en iyi degeri sakla
                best_obj = min(obj_all)
                # en dusuk yanlisa sahip degeri sakla
                if best_obj < prev_obj:
                        best_vector = pop[argmin(obj_all)]
                        prev_obj = best_obj
                        # her yineleme sonrasi kotrol edip degerleri degistir
        return best_vector
```

Kullanıcı tarafında Future Selection işlemi yapılacak veri setinin girişinin yapılması için bir kod yazılmıştır. Burada kullanıcı istediği veri setinin ismini girerek işlemin başlatılmasını sağlar.

```
# datayi oku
try:
        data  = pd.read_csv(input('Veriseti dosya ismini giriniz: '))
        # data  = pd.read_csv('test.csv')
        veriler  = data.values
except:
        print("Dosya okunurken sorun olustu.")
        exit(1)

print("Islemler baslatildi.")
```

Burada Diferansiyel Evrim Algoritması içerisinde kullanılcak olan parametreler kullanıcı tarafından belirlenmekte ve yazılmaktadır.

```
# pop boyutu
pop_boyutu = 20
# iterasyon sayisi
iterasyon_carpani = 1000
# mutasyon carpani
mutasyon_carpani = 0.2
# caprazlama carpani
cc = 0.9
```

Diferansiyel Evrim Algoritması yukarıda bir definasyon olarak çalışmaktadır. Bu sebep ile burada o definasyonu çağırmakta ve çalıştırmaktayız.

```
# diferansiyel evrimi calistir
try:
        solution = diferansiyel_evrim(pop_boyutu, veriler, iterasyon_carpani, mutasyon_carpani, cc)
except:
        print("Diferansiyel evrim yapilirken sorun olustu.")
        exit(1)
```

Seçilen sonuçlar içerisinde Null veriler bulunuyorsa onları sonuç kısmından çıkarıyoruz ve kalanlardan kaç tane değer seçildiğini görüyoruz.

```
# gelen sonuc secindeki sifir olmayan alanlari alip, boyutunu ekrana yazdiriyoruz
solution = solution[solution != 0]
print("Sonuc sayisi:", len(solution))
```

```
Sonuc sayisi: 32
```