

**EGE ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
2019 – 2020 ÖĞRETİM YILI**



**YAPAY ZEKA YÖNTEMLERİ VE
UYGULAMALARI DERSİ
PROJE 1 – RAPORU**

Ege Doğan Dursun
05170000006

1. Algoritmalar, Tanımlar, Karşılaştırma, Araştırma ve Yorum

1.1. Kalemle yazılan algoritmaların tarayıcı görüntüleri veya fotoğrafları

05170000006
Ege Doğan DURSUN
Yapay Zeka Yöntemleri
ve
Uygulamaları
Ege Üniversitesi 2019-2020
Bahar Dönemi
PROJE - 1

Sayfa 1

1 SORU

a) Tepe Tırmanma (Hill Climbing) Algoritması

i) Lokal maksimum: Komşulardan daha iyi bir state'i belirtir, ancak uzayda daha iyi bir state mevcuttur.

ii) Global maksimum: Durum uzayındaki en iyi state olan en iyi state'i belirtir. Objective Function'in en yüksek değeriidir.

iii) Mevcut state: Durum uzayında mevcut halde bulunan durundur.

iv) Düz lokel maksimum: Durum uzayının komşuları ile aynı değer taşıyan konumur.

v) Omuz: Yukarı eğim taşıyan bir köşest olan plato bölgesidir.

Objectice Function
↑
↓ State space

Simple Hill Climbing Algoritması

Adım 1: Mevcut state'i değerlendir. Amasların state bu ise dur.

Adım 2: Bir çözüm bulunana kadar dön, ya da uygulanacak operator kalmadına kadar dön.

Adım 3: Mevcut state için bir operator succ ve ugula.

Adım 4: Yeni state'i kontrol et.

- Eğer amasların state ise dur.
- Mevcut state'den daha iyi ise yeni state'i mevcut state olarak seç.
- Eğer mevcut state'den daha iyi değilse (Adım 2) ye dön.

Adım 5: Dur

Steepest-Ascent Hill Climbing Algoritması

Adım 1: Mevcut state'i değerlendir. Eğer amasların state bu ise dur, yoksa beslenen state'ini mevcut state olarak belirle.

Adım 2: Sonuç bulunana kadar mevcut state değişmeyeceğine kadar dön:

- succ öyle bir state olsun ki, mevcut state'in her yenisinden daha iyi olsun.
- Mevcut state'e uygulanabilecek her operator teh:
 - Operator'u ugula ve yeni bir state oluştur.
 - Yeni state'i değerlendir.
 - Amasların state bu ise dur, yoksa succ ile karşılaştır.
 - succ den daha iyi ise, succ'ye yeni state'i atla.
 - succ den daha iyi ise, mevcut state'i succ olarak değiştir.

Adım 3: 0

05170000006

Ege Dijital Mühendislik

Yapay Zeka Temelinde

ve

Uygulamaları

Ege Üniversitesi 2019-2020

Bölüm: Dijital

PROJE-1

Sayı-2

1.5.0.RÜ

a) Isıl İstem (Simulated Annealing) Uygulaması

// Pseudocode

```
initialize (temperature T, random starting point)
while cool_iteration <= max_iterations
    Cool_iteration = cool_iteration + 1
    temp_iteration = 0
    while temp_iteration <= nrep
        temp_iteration = temp_iteration + 1
        Select a new point from the neighborhood
        Compute current_cost (of this point)
        δ = current_cost - previous_cost
        if δ < 0, accept neighbour
        else, accept with probability exp(-δ/T)
    end while
    T = α * T (0 < α < 1)
end while
```



— Tepe Tırmanma (Hill Climbing) Algoritma Kompleksitesi = {Time Complexity: $O(\infty)$, Space Complexity: $O(b)$ }

★ Birçok problem için polinom zamanla gizemle edilebilir.

★ NP-complete problemler için lokal maksimuma bağlı olarak eksponansiyel büyüme olabilir.

— Simulated Annealing (Isıl İstem) Algoritma Kompleksitesi = {Time Complexity: $O(n^2) \rightarrow O(n)$, Space Complexity: $O(n^2)$ }

★ Time Complexity is $O(n^2)$ for initial stage,

then becomes linear ($O(n)$) with each iteration. (slideplayer.com/slide/9898074/)

maddde 12

★ NP-complete ve NP-hard Problemlerde eksponansiyel büyüme gorulebilir.

Éric Dognin DURSUN

Yanın tere Jidemki

see *Myrmecole*.

Foto: Anversidad 2019-2020

Q98N-1

b) Tanım ve Korsılastırma (1.soru)

Açıklayınız:

* Admissible Heuristic (Murdeler sergisi) bir algoritmadır.

$$f(n) = g(n) + h(n)$$

↑	↑	↑
hespinne yepn segzel fekysen	Boligang digunah mercat digune gebran maligall	Mercum digunah tedel digune keda gelowas tedeh maligall

ALGORITMA:

① Her adımda en düşük değer olan (en öreni) düşüm alınır. (Bu düşüme gidiyor.) ve düşüm sıradańa girerilir.

② Gördiğimizdeki gibi konusunda birin
değerlerini değerlendirmek (Artık bu değerler
şartnameye uygun olup olmadığı) konusunu
kullanılarak yapabiliriz.

③ Algoritmo que se da en la actividad anterior. Se detallan los pasos:

0517000006
 Ege Nefes Dersi
 (Topluk Zeka Yontemleri)
 Xe
 Uygulamalar
 Ege Universitesi 2019-2020
 Bahar Nötemi
 PROJE - 1

Sayfa 4

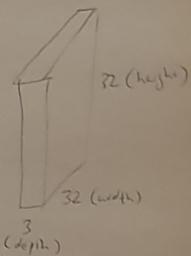
1. soeu

b) Tanim ve Kasilistirme

Aitkenin:

III) Convolutional Neural Networks (CNN): Tipki klasik yapay sinir aglari gibi, CNN'le de celsili blaster ve degisirilebilir / ogrenebilir aqisillikla satip noronlardan meydana gelir. Her noron celsili sayide input olur, bu inputların egitsiz toplami hesaplar Lee ordundan altivasyon fonksiyonundan geçirir. Böyledice bir output ile cekop verir. Tum sinir agi din loss fonksiyonuna sahiptir. [3]

* Yapay sinir aglari CNN'ler vectorler yeme yetki kazanmış goruntuler isleyebilirler.



Convolution



* Goruntiden $5 \times 5 \times 3$ filtre yardimciyle kesişte alınıcak ve bu sadece filtre ve input goruntusunun parselinin dot productu hesaplanır.

* Her dot product'un gorun stoke izlenmektedir.



* Sonra next olası her pozisyonda alınan parselin dot productlarının toplamı elde edilir.



* Örneğin 6 farklı filtre kullanılarak CNN'de üst usk stocklenen goruntülerden 6 adet feature map elde eder (Bu bir 28x28x1)

Pooling



* Specified size'li kütüklerde sinir aglara hizli bir parametre saglama gerekliyor. En yeggin kütüklerin yontem "Max pooling" tir.

1	1	2	4
2	(6)	7	8
3	2	1	0
1	2	7	(4)

max pool with
2x2 filter and stride 2

6	8
3	4

Efe Dögez DURSUN

0517000006
Ege Doğa DÜZÜN
Yapı Telsiz Teknolojileri
ve Uygulamaları
Ege Üniversitesi 2019-2020
Zeka Dersi
PROJE-1

Sayfa 6

1. SORU

c) Veri bilimi alanından başlayarak, kognitif genetik hatalar.

- ① Teorinin içerişine çok fazla dalmış, pratig'e his saldırmak: Teorinin keşfedilmesi bu alanın başlıca olundurması konusundan önce bunun pratikle dengeli bir uygulama içinde gerçekleştirilmesi en sevildisidir.
- ② Genetikteki ve özniteliklerin öğrenmeden algoritmalarla deinfenesine dair: Öncelik modeldeki bağısalılığı gibi teori ve pratigin uygun bir dengede ilişkilendirme genetikteki Öğrenimdeki bir konum pratigin yapısını toraklı olduğu gibi, öğrenilen bir konum pratikle pekiştirilmeli de sevildir.
- ③ Direkt öğrenmeye baslondu: Birin öğrenme motivağı öğrenme, yapısal telsiz ve veri bilim araclarından sadece bir alt kümeli. Tercih motivağı öğrenme teorisini kez veri bilim ilkelemi keşfetmek sonucunda derin öğrenmeye uygunlaşır.
- ④ Accuracy'ı osrularak modelin nasıl çalıştığını esitmek: Accuracy her zaman dengeli, doğaldan bağımsızdır.
- ⑤ Müstak teknikte çok fazla teknik kullanmak: Konsantre insansın ürettiği modelin teknikinde doğal, doğaldan bağımsızdır.
- ⑥ Bir teknoloji birinden fazla arası teknoloji, dil öğrenmeye çalışmak: Eğer bu koda fazla arası teknoloji varsa, bu teknoloji birinden fazla teknoloji kullanımda kullanılmamalıdır.
- ⑦ Probleme kafa yormadan kod yazmak: Önce problemi onaltı editleş, buna uygun aday çözümleri ürettilmesi, uygun bir model belirlenmesi gereklidir. Kod ve uygulama aynı anda yapılmalıdır, en sonunda kendi içinde uygulanmasını elde etmesi gereklidir.
- ⑧ Direkt Çalışma: Veri bilimi gen bir alanın ve öğrenilebilir konu obiecti, doğa ve teknik genetiktedir. Bu yalanı doğal olarak kısaltır.
- ⑨ Aşırı uygun ve aşırı düşük hatalı: Bu da Accuracy'ın katışı teknik ile doğal hataları bir hale getirir. Verilen durum göre gereklilığından onaltı editleş, gerçekte düşündürmek istenilen modelin 99,999 accuracy sağlarken bile hemen neyin yanlışlığını anlatır.

1.2 Tanım ve Karşılaştırmalar (Bilgisayarda yazılabilir)

NOT: Yukarıdaki görsellerde soruların cevapları bulunmaktadır. Okunurluğun düşük olabilecek olması düşüncesiyle yazılıanları buraya da geçiriyorum.

A* Algoritması : Bilgisayar bilimlerinde en kısa yol bulmak için kullanılan algoritmaların birisidir. Örneğin; Traveling Salesman Problem çözümünde kullanılabilir. Benzer şekilde oyun programlamada, oyuncuların en kısa yolu bularak hedefe gitmeleri için de sıkılıkla kullanılan algoritmadır. [1]

Kısaca bir node'dan başka bir hedef node'a en kısa hangi düğümlerin kullanılarak gidilebileceğini bulmaya çalışan bir best fit algoritmasıdır.

Admissible Heuristic (Muteber Sezgisel) bir algoritmadır.

$$f(n) = g(n) + h(n)$$

f(n) : hesaplama yapan sezgisel fonksiyon

g(n) : başlangıç düğümünden mevcut düğüme gelmenin maliyeti

h(n) : mevcut düğümden hedef düğüme gelmenin tahmini maliyeti

Algoritma:

1. Her adımda en düşük değeri olan (en önemli) düğüm alınır. (Bu düğüme gidilir) ve düğüm sıradan çıkarılır.
2. Gidilen düğüme göre komşu olan bütün düğümlerin değerleri güncellenir. (Artık bu düğüme gitmenin bir maliyeti vardır ve f(n) fonksiyonu içerisinde bu yer alır.)
3. Algoritma yukarıdaki adımları hedefe varana kadar, (hedef düğüm priority queue'da en öne gelene kadar) tekrarlar veya sırada düğüm kalmayana kadar tekrarlar.

Auto ML: Automated Machine Learning (AutoML) uçtan uca makine öğrenmesi (ML) sürecini uygulamaktır ve amacı işletmelerin faydalananabileceği tahminleyici sistemler yaratmak, geliştirmek ve faaliyete koymaktır. AutoML araçları, veri bilimcilere tahminleyici sistemler geliştirmenin uçtan uca yaşam döngüsünü otomatize etmek açısından fayda sağlar. Örneğin veri hazırlama, veri seçme, model ayarlama, eğitim, doğrulama, faaliyete sokma ve yönlendirme. Özellikle iş problemlerinin boyutu büyükçe ve karmaşıklığı arttıkça, daha çok makine öğrenmesi sistemine ihtiyaç duyulmaktadır ve bu yüzden veri bilimi takımlarının verimliliklerini artırmak için AutoML sistemleri kullanılmaktadır. [2]

Convolutional Neural Networks (CNN) : Tıpkı klasik yapay sinir ağları gibi, CNN'ler de çeşitli biaslar ve değiştirilebilir/öğrenebilir ağırlıklara sahip nöronlardan meydana gelirler. Her nöron çeşitli sayıda input alır, bu inputların ağırlıklı toplamını hesaplar ve ardından aktivasyon fonksiyonundan geçirir. Böylece bir output ile cevap verir. Tüm sinir ağı bir loss fonksiyonuna sahiptir. [3]

Yapay sinir ağlarının aksine CNN'ler vektörler yerine çok kanallı görüntülerini işleyebilirler.

Convolution :

- Örneğin en:32 , boy: 32 , derinlik : 3 'lük bir görüntü var.
- Görüntüden $5 \times 5 \times 3$ filtre yardımıyla kesitler alınır ve bu sırada filtre ve input görüntüsünün parçalarının dot product'ı hesaplanır.
- Her dot product için çözüm skaler özellikleştir.
- Sonuç olarak olası her pozisyondan alınan parçaların dot product'larının birleştirilmesiyle ortaya $28 \times 28 \times 1$ lik bir görüntü çıkar.
- Örneğin 6 farklı filtre kullanılan bir CNN'de üst üste stack'lenen görüntülerden 6 adet feature map ortaya çıkar. (Her biri $28 \times 28 \times 1$)

Pooling :

- Spatial Size'ı küçülterek sinir ağlarında hesaba katılacak parametre sayısını azaltmaya çalışır. En yaygın kullanılan yöntem "Max Pooling".

Random Forest vs. Decision Tree : Random Forest özünde Decision Tree'lerin bir koleksiyonudur. Tüm dataset üzerine bir Decision tree oluşturulur, bu esnada ilgilenilen featurelar/değişkenler kullanılır. Random Forest ise rastgele gözlemler/satırlar ve spesifik featurelar/değişkenler seçerek fazla sayıda Decision tree oluşturur ve sonuçların ortalamasını alır. Bu yöntemde yüksek sayıda ağaç oluşturulduktan sonra her ağaç "oylama" yoluyla sınıf seçer ve en yüksek oyu alan sınıf tahminlenen sınıf olur. [4]

Strong AI vs. Weak AI : Weak AI, zaten bildiği konularda çeşitli sensörel, input verilerini değerlendирerek uygun sınırlandırmalar yapar. Bu her ne kadar insansı bir davranış tipi gibi görünse de aslında Weak AI , kavramsal anlayıştan uzaktır. Sınıflandırma konsepti üzerinde değer taşıyan parametreleri kullanır ve kapsamlı bir anlayış güdemez. Örneğin, SIRI'ye "klimayı aç" derseniz, burada klima ve aç kelimelerini bağdaştırarak klimayı açar ancak sözü geçen cümle ile ilgili kavramsal bir anlayış sahibi değildir. Strong AI ise daha unsupervised ve insan beynine yakın şekilde çalışan AI'dır. Basit sınıflandırmalar değil, insan beyni gibi çıkarımlar ve clustering'e dayanır. Dolayısı ile aslında durumlara karşı verilen programlı cevaplardan değil, stokastik ve daha önceden tahminlenemez, kavramsal anlayış sonucu oluşan cevaplarla karşılaşırız. Weak AI, Strong AI ile kıyasla insan müdahalesine daha çok bağımlıdır. [5]

Atıflar:

- [1] Şeker, Sadi Evren. (Mart 2, 2009). A* Arama Algoritması.
<http://bilgisayarkavramlari.sadievrenseker.com/2009/03/02/a-yildiz-arama-algoritmasi-a-star-search-algorithm-a/>
- [2] <https://www.tazi.ai/auto-ml/>
- [3] Pokharna, Harsh. (Temmuz 28, 2016). The Best Explanation of Convolutional Neural Networks on the Internet!
<https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8>
- [4] Stats.Student. (Haziran 17, 2017).
<https://stats.stackexchange.com/questions/285834/difference-between-random-forests-and-decision-tree>
- [5] E-3 Magazine. (Mayıs 27, 2019). Strong vs. Weak Artificial Intelligence.
<https://e3zine.com/strong-artificial-intelligence/>

1.3 Araştırma ve Yorum

NOT: Yukarıdaki görsellerde soruların cevapları bulunmaktadır. Okunurluğun düşük olabilecek olması düşünücsiyile yazılınları buraya da geçiriyorum.

1. Teorinin içeresine çok fazla dalıp, pratiğe hiç odaklanmamak. Teorinin kavranması bu alanda başarılı olunabilmesi için elzemdir ancak bunun pratikle dengeli bir uyum içerisinde gerçekleştirilmesi en sağılıklıdır.
2. Gereksinimleri ve ön kaynakları öğrenmeden algoritmala derinlemesine dalmak. Önceki maddede de bahsedildiği gibi teori ve pratiğin uygun bir dengede ilerletilmesi gerekmektedir. Öğrenilmeyen bir konunun pratiğini yapmak zararlı olduğu gibi, öğrenilen bir konunun pratikle pekiştirilmemesi de zararlıdır.
3. Direk olarak derin öğrenmeye başlamak. Derin öğrenme; makine öğrenmesi, yapay zeka ve veri bilimi araçlarından sadece bir alt kümedir. Temel makine öğrenmesi teorisini ve veri bilimi ilkelerini kavramadan önce derin öğrenmeyle uğraşmak yanlıştır.
4. Accuracy değerine aşırı odaklanarak modelin nasıl çalıştığını es geçmek. Accuracy her zaman her şeyi ölçmez. Belli modellerde %70 accuracy oranı harikayken, bazlarında %95 kötü olabilir. Modelin detayları ve kapsamı gözden kaçırılmamalıdır.
5. Müşteri iletişiminde çok fazla terim kullanmak. Karşınızdaki insana ürettiğiniz modelin tekniqinden ve teknik detaylarından değil, faydalardan bahsetmelisiniz.
6. Bir defada birden fazla araç, teknoloji, dil öğrenmeye çalışmak. Eğer bu kadar fazla noktaya aynı anda bölünmeye çalışırsanız, en sonunda kendinizi hiçbirinde uzmanlaşmamış olarak bulursunuz.
7. Probleme kafa yormadan kod yazmak. Önce problemin analiz edilmesi, buna uygun aday çözümlerin üretilmesi, uygun bir model belirlenmesi gereklidir. Kod ve uygulama ancak bu tarz bir beyin firtınası sonrasında gerçekleşmelidir.
8. Düzenli çalışmamak. Veri bilimi geniş bir alandır ve öğrenebilmek için düzenli çaba, disiplin ve pratik gerekmektedir. Bu yüzden düzenli olarak vakit ayrılmalıdır.
9. Aşırı uyum ve az uyumu hafife almak. Bu da accuracy değerine aşırı kafayı takmak ile bağlantılı bir hata olabilir. Verinin duruma göre güvenilirliğinin analiz edilmesi gereğinin, gerçek dünyadan uzak bir modelin %100 accuracy sağlasa bile hiçbir işe yaramayacağının unutulmaması gereklidir.

2. “Bridge and Torch” Problemini A* Yöntemi ile Çözen Program (2. Alternatif)

2.1. Problemin Tanımı

Bridge and Torch Problem : “*N kişi* gece bir nehrin kıyısına geliyor. Dar bir köprü var, ve bu köprü aynı anda ancak 2 kişiyi kaldırabiliyor. 1 adet *meşaleleri* var, ve gece olduğu için köprüyü geçerken bu *meşaleyi* kullanmaları gerekiyor. Her *kışının* köprüyü geçme süresi birbirile aynı ya da farklı olmak ile birlikte $X(i)$ dakika sürüyor. Eğer köprüden 2 kişi aynı anda geçiyorsa, bu geçiş yavaş olan *kışının* geçiş hızı kadar sürüyor. Soru, *meşalenin* Y dakika boyunca yandığı göz önüne alınırsa, herkesin köprüden geçmeyi başarması mümkün müdür.

N : Kişi Sayısı

$X(i)$: *i* kişisinin geçiş hızı

Y : Meşalenin yanma süresi

Aktörler:

1. Kişi / Kişiler
2. Köprü
3. Meşale

2.2. Programın Özellikleri (Kullandığı Yöntem ve Mekanizmalar, Ekstraları, Orjinallikleri)

Programda öncelikli olarak Python dili kullanarak Object Oriented Programming prensiplerine uygun bir şekilde, probleme has aktörleri tanımlayan gerekli sınıfları yarattım.

Bu sınıflar; Bridge, Person ve Torch sınıfları.

Bridge : Bu sınıf problemimizi tanımlayan ana objeyi esas alır. Constructor içerisinde köprüyü geçmek isteyen insanları içeren bir liste olan people değişkenine sahip şekilde oluşturulur.

Person : Bu sınıf problemimizin ikinci ana aktörü olan “*kışileri*” tanımlama amacıyla yaratılmış sınıfıtır. Kişiler geçiş sürelerini ifade eden pass_time değişkenine sahip şekilde yaratırlar.

Torch : Problemimizi tanımlayan son ana aktörü olan *meşaleyi* tanımlayan sınıfıdır. *Meşaleyi* meydana getiren ana özellik de onun yanlış süresini ifade eden burn_time değişkenidir. *Meşale* objelerinin örnekleri, bu değişkene sahip biçimde yaratılırlar.

Programda Kullanılan Diğer Sınıflar:

ASharp_Solver : Bu sınıf probleme dahil olan aktörlerle ilgili gerekli hesaplamaları yaparak ilgili çözümü üreten metotları barındıran sınıfır. Bu sınıfa dahil olan metotları aşağıdaki gibi inceleyebiliriz:

visualize_bridge():

-parametreler : start_edge (dict) , end_edge(dict)
-return : void

Bu metot çözüm süresinde köprünün baş ve son uçlarındaki kişi durumunun kullanıcıya daha etkin bir biçimde gösterilmesini sağlamak amacıyla dizayn edilmiş basit bir print metodudur.

```
Total People Time : 0  
[START-EDGE] ( C ) ( D ) ( E ) ( ) ( ) / ======BRIDGE===== / ( ) ( ) ( ) ( A ) ( B ) [END-EDGE]  
Total People Time : 1
```

Görüntünün sol tarafında köprünün başlangıç ucunu ve ucta yer alan kişileri, sağ tarafında ise köprünün bitiş ucunu ve bu ucta yer alan kişileri gözlemleyebiliriz.

show_problem_definition():

-parametreler : bridge (Bridge) , torch (Torch)
-return : void

Bu metot kullanıcıya problemin çözümüne başlamadan önce, problemle ilgili kısa bir bilgilendirmede bulunmayı ve problemi oluşturan ana aktörlerle ilgili bilgi vermeyi amaçlayan basit bir print metodudur.

```
BRIDGE AND TORCH PROBLEM DEFINITION  
'N' PEOPLE come to a river in the night. There is a narrow BRIDGE, but it can only hold '2' PEOPLE at a time.  
They have 1 torch and, because it's night, the TORCH has to be used when crossing the BRIDGE.  
Each PERSON can pass the BRIDGE in 'X' minutes. When '2' PEOPLE cross the BRIDGE together, they must move at the 'SLOWER PERSONS'  
pace.  
The question is, can they all get across the BRIDGE if the TORCH lasts only 'Y' minutes?
```

```
Total PEOPLE Amount : 5  
Burn Time of the TORCH : 30  
PEOPLE Information :  
PERSON A can pass the bridge in 1 minutes.  
PERSON B can pass the bridge in 3 minutes.  
PERSON C can pass the bridge in 6 minutes.  
PERSON D can pass the bridge in 8 minutes.  
PERSON E can pass the bridge in 12 minutes.
```

solve_bridge_and_torch_assharp():

-parametreler : bridge (Bridge), torch (Torch)
-return : Void

Bu metot problem tanımlama ile ilgili fonksiyonun çağrılmasını ve A* Algoritması kullanılarak problemin çözümünün araştırılmasını sağlayan metodu çağrıma görevi üstlenen bir wrapper/kılıf metottur.

solve_bridge_and_torch() :

-parametreler : bridge(Bridge), torch(Torch), n_epochs(int)

-return : satisfied (Boolean), tour_count(int)

Bu metot problem tanımlama ile ilgili fonksiyonun çağrılmasını ve Brute Force kullanılarak problemin çözümünün n_epochs ile tanımlanan tur sayısı içinde araştırılmasını sağlayan bir wrapper/kılıf metottur.

get_min_pass_time() :

-parametreler : people (dict)

-return : min_name(String), min_person(Person)

Bu metot, bir grup insan içerisinde cost'u en düşük olan kişiyi, yani başka bir deyişle köprüyü geçme süresi en düşük olan kişiyi alan yardımcı bir metottur.

determine_route_assharp() :

-parametreler : bridge (Bridge), torch (Torch)

-return : void

Bu metot problemin çözümünü A* Algoritması kullanarak çözen esas metottur. Bu metotun içerisinde gerçekleştirdiği faaliyeti özetleyecek olursak, köprüdeki her gidiş serüveni için mevcut kişiler grubu içerisindeki gidiş maliyeti en düşük olan birinci ve ikinci kişiyi tercih ederken, her dönüş serüveni için köprünün bitiş ucundakiler arasından dönüşü en az maliyetli olan kişiyi tercih ederek ilerlemektedir. Bu sayede gidiş ve gelişlerin ardından, herkes karşıya geçmeyi başardığında oluşan toplam maliyetin optimize edilerek kabul edilebilir bir seviyeye taşınması amaçlanır. Ancak her zaman meşale limitinin altında bir sonuca erişilmesi garanti edilmez.

determine_routes() :

-parametreler : bridge (Bridge), torch(Torch)

-return : satisfied (Boolean)

Bu metot problemin çözümünü Brute Force kullanarak çözmeye çalışan esas metottur. A* algoritması gibi optimize edilmiş, kabul edilebilir düşük bir sonuç bulmaktan ziyade, meşalenin yanlış süresinin altında olan; tam anlamıyla uyumlu bir cevap bulmayı amaçlar ancak bunun gerçekleşmesinin zamanı konusunda bir süre garanti etmez. Her turda karşıya geçecek olan rasgele iki kişi seçilir ve karşıya geçirilir. Ardından her dönüş turunda rasgele bir kişi seçilir ve geri dönüşü sağlanır. Eğer geçen toplam süre meşalenin yanlış süresinin üzerine çıkarsa epoch başarısız kabul edilir, altında veya aynı olursa başarılı kabul edilir. Başarı durumunu tanımlayan satisfied değişkeni döndürülür.

Test Sınıfı:

Bu sınıfın içinde öncelikle problemimizle ilgili gerekli aktörler özellikleri ile yaratılır. Program dahilinde 2 farklı problem tanımı gerçekleştirdim. Bunlar:

N : insan sayısı, X(i) : kişinin geçiş süresi , Y : meşalenin toplam yanma süresi olmak üzere;

1. Problem:

N : 4 adet insan

X(A) : 1 dakika

X(B) : 2 dakika

X(C) : 5 dakika

X(D) : 8 dakika

Y : 15 dakika

2. Problem:

N : 5 adet insan

X(A) : 1 dakika

X(B) : 3 dakika

X(C) : 6 dakika

X(D) : 8 dakika

X(E) : 12 dakika

Y : 30 dakika

NOT: Problemin esas çözümlemesini ve gerçekleştirdiğim hesaplara ek olarak raporuma belirttiğim çıktıları "Problem 2" esas alarak gerçekleştirdim. Bunu seçmemde ana sebep, ödev dökümanında verilmiş olan linkteki problemin bu şekilde tanımlanmış olmasıdır.

İlgili problem tanımı gerçekleştirildikten sonra, problemin A* Algoritması kullanılarak çözülmesi için gerekli olan fonksiyon ilgili parametreler kullanılarak çağrılır. Çözüm kullanıcıya gösterilir.

Bu noktada bahsetmek istediğim temel gözlem, A* Algoritmasının söz konusu problemin çözümünde optimize edilmiş bir geçiş süresi sağlamak ile birlikte meşalenin toplam yanlış süresinin altında kalacak bir süreyi sağlamakta başarısız olmasıdır. Bu duruma karşı çözüm olarak, olası gerçek bir çözümün varlığını sağlamakla görevlendirilen ve Brute Force yöntemi ile çalışan ek bir metot geliştirdim. Bu metot problemin çözümü konusunda bir zaman garantisini vermemek ile birlikte problemin mevcut çözümlerini belli bir tur sayısında arayışta bulunma amacıyla güder.

Test modülünün ilerleyen bölümlerinde Brute Force algoritması devreye sokularak toplamda “25” kez ilgili problemin çözülmesi sağlanır. Bu çözümler sırasında her çözüm periyodunun kaç saniye sürdüğü ve kaç epoch boyunca araştırma yapıldığı kaydedilir. Ardından bu metrikler ve buna ek olarak ortalama araştırma süresi ve ortalama epoch sayısı da kullanıcıya gösterilir.

2.3. Programın Ekran Görüntüleri

Kod Görselleri:

```
#Importing the necessary libraries and modules
from Person import Person
from Torch import Torch
from Bridge import Bridge
from ASharp_Solver import solve_bridge_and_torch, solve_bridge_and_torch_asharp
import time as t

#Defining people for the first type of problem
people_1 = {
    "A" : Person(pass_time=1),
    "B" : Person(pass_time=2),
    "C" : Person(pass_time=5),
    "D" : Person(pass_time=8),
}

#Defining people for the second type of problem
people_2 = {
    "A" : Person(1),
    "B" : Person(3),
    "C" : Person(6),
    "D" : Person(8),
    "E" : Person(12),
}

#Defining the torches for the problem
torch_1 = Torch(burn_time=15)

torch_2 = Torch(burn_time=30)

#Defining the Bridges we will use for the problem
bridge_1 = Bridge(people=people_1)

bridge_2 = Bridge(people=people_2)
```

```

#This is commented out because it belongs to the question 1, which is another version of the problem, |
#that was mentioned on the homework file.
"""
result_1, _ = solve_bridge_and_torch(bridge=bridge_1, torch=torch_1, n_epochs=1000)
print("Question 1 : Is there a result? : ", ("YES" if result_1 else "NO"))
"""

#Solving the problem with A* algorithm
solve_bridge_and_torch_asharp(bridge_2, torch_2)

```

```

#Solving the question N number of times
times = []
epoch_to_solution = []
for i in range(0, total_sol_num):
    start = t.time()
    result_2, tour_count = solve_bridge_and_torch(bridge=bridge_2, torch=torch_2, n_epochs=5000)
    end = t.time()
    tot_time = end-start
    print("Question 2 : Is there a result? : ", ("YES" if result_2 else "NO"))
    times.append(tot_time)
    epoch_to_solution.append(tour_count)

#Show the times until finding the solution for each tour.
for i in range(0, len(times)):
    print("For Solution Number ", (i+1), " -> Time until finding the solution : ", times[i], " seconds.")

print("\n")

#Show the average time until finding the solution
avg_time = sum(times) / len(times)
print("AVERAGE TIME TO FIND THE SOLUTION: ", avg_time, " seconds.")
print("\n\n")

#Show the epochs until finding the solution for each tour.
for i in range(0, len(epoch_to_solution)):
    print("For Solution Number ", (i+1), " -> Number of epochs to find the solution : ", epoch_to_solution[i])

print("\n")

#Show the average epoch amount until finding the solution
avg_epoch = sum(epoch_to_solution) / len(epoch_to_solution)
print("AVERAGE EPOCH AMOUNT TO FIND THE SOLUTION: ", avg_epoch)

```

Kod Çıktıları:

```
In [12]: solve_bridge_and_torch_asharp(bridge_2, torch_2)
BRIDGE AND TORCH PROBLEM DEFINITION
'N' PEOPLE come to a river in the night. There is a narrow BRIDGE, but it can only hold '2' PEOPLE at a time.
They have 1 torch and, because it's night, the TORCH has to be used when crossing the BRIDGE.
Each PERSON can pass the BRIDGE in 'X' minutes. When '2' PEOPLE cross the BRIDGE together, they must move at the 'SLOWER PERSONS' pace.
The question is, can they all get across the BRIDGE if the TORCH lasts only 'Y' minutes?
```

```
Total PEOPLE Amount : 5
Burn Time of the TORCH : 30
PEOPLE Information :
PERSON A can pass the bridge in 1 minutes.
PERSON B can pass the bridge in 3 minutes.
PERSON C can pass the bridge in 6 minutes.
PERSON D can pass the bridge in 8 minutes.
PERSON E can pass the bridge in 12 minutes.

[START-EDGE] ( A ) ( B ) ( C ) ( D ) ( E ) /=====BRIDGE=====/ ( ) ( ) ( ) ( ) ( ) [END-EDGE]
Total Elapsed Time : 0
[START-EDGE] ( C ) ( D ) ( E ) ( ) ( ) /=====BRIDGE=====/ ( ) ( ) ( ) ( A ) ( B ) [END-EDGE]
Total Elapsed Time : 3
[START-EDGE] ( C ) ( D ) ( E ) ( A ) ( ) /=====BRIDGE=====/ ( ) ( ) ( ) ( ) ( B ) [END-EDGE]
Total Elapsed Time : 4
[START-EDGE] ( D ) ( E ) ( ) ( ) ( ) /=====BRIDGE=====/ ( ) ( ) ( B ) ( A ) ( C ) [END-EDGE]
Total Elapsed Time : 10
[START-EDGE] ( D ) ( E ) ( A ) ( ) ( ) /=====BRIDGE=====/ ( ) ( ) ( ) ( B ) ( C ) [END-EDGE]
Total Elapsed Time : 11
[START-EDGE] ( E ) ( ) ( ) ( ) ( ) /=====BRIDGE=====/ ( ) ( B ) ( C ) ( A ) ( D ) [END-EDGE]
Total Elapsed Time : 19
[START-EDGE] ( E ) ( A ) ( ) ( ) ( ) /=====BRIDGE=====/ ( ) ( ) ( B ) ( C ) ( D ) [END-EDGE]
Total Elapsed Time : 20
[START-EDGE] ( ) ( ) ( ) ( ) ( ) /=====BRIDGE=====/ ( B ) ( C ) ( D ) ( A ) ( E ) [END-EDGE]
Total Elapsed Time : 32
```

(A* Algoritması kullanılarak problemin çözümlenmesi, sonuç olarak 32 dakikada çözüme ulaşılmış.)

```
[START-EDGE] ( A ) ( D ) ( E ) ( C ) ( ) /=====BRIDGE=====/ ( ) ( ) ( ) ( ) ( B ) [END-EDGE]
[START-EDGE] ( A ) ( E ) ( ) ( ) ( ) /=====BRIDGE=====/ ( ) ( ) ( ) ( B ) ( C ) ( D ) [END-EDGE]
[START-EDGE] ( A ) ( E ) ( C ) ( ) ( ) /=====BRIDGE=====/ ( ) ( ) ( ) ( ) ( B ) ( D ) [END-EDGE]
[START-EDGE] ( A ) ( ) ( ) ( ) ( ) /=====BRIDGE=====/ ( ) ( ) ( B ) ( D ) ( E ) ( C ) [END-EDGE]
[START-EDGE] ( A ) ( E ) ( ) ( ) ( ) /=====BRIDGE=====/ ( ) ( ) ( B ) ( D ) ( C ) [END-EDGE]
Failed! Total Elapsed Time : 50

Total Solution Searches : 69
[START-EDGE] ( A ) ( B ) ( C ) ( D ) ( E ) /=====BRIDGE=====/ ( ) ( ) ( ) ( ) ( ) [END-EDGE]
[START-EDGE] ( A ) ( B ) ( D ) ( ) ( ) /=====BRIDGE=====/ ( ) ( ) ( ) ( E ) ( C ) [END-EDGE]
[START-EDGE] ( A ) ( B ) ( D ) ( C ) ( ) /=====BRIDGE=====/ ( ) ( ) ( ) ( ) ( E ) [END-EDGE]
[START-EDGE] ( A ) ( B ) ( ) ( ) ( ) /=====BRIDGE=====/ ( ) ( ) ( E ) ( C ) ( D ) [END-EDGE]
[START-EDGE] ( A ) ( B ) ( D ) ( ) ( ) /=====BRIDGE=====/ ( ) ( ) ( ) ( E ) ( C ) [END-EDGE]
Failed! Total Elapsed Time : 34

Total Solution Searches : 70
[START-EDGE] ( A ) ( B ) ( C ) ( D ) ( E ) /=====BRIDGE=====/ ( ) ( ) ( ) ( ) ( ) [END-EDGE]
[START-EDGE] ( A ) ( B ) ( C ) ( ) ( ) /=====BRIDGE=====/ ( ) ( ) ( ) ( E ) ( D ) [END-EDGE]
[START-EDGE] ( A ) ( B ) ( C ) ( E ) ( ) /=====BRIDGE=====/ ( ) ( ) ( ) ( ) ( D ) [END-EDGE]
[START-EDGE] ( A ) ( B ) ( ) ( ) ( ) /=====BRIDGE=====/ ( ) ( ) ( D ) ( E ) ( C ) [END-EDGE]
[START-EDGE] ( A ) ( B ) ( E ) ( ) ( ) /=====BRIDGE=====/ ( ) ( ) ( ) ( D ) ( C ) [END-EDGE]
Failed! Total Elapsed Time : 48

Total Solution Searches : 71
```

(Brute Force yöntemi kullanılarak probleme ait çözüm arama süreci ve başarısız olunmasına sebep olunan toplam süre değerleri)

```
Total Solution Searches : 110
[START-EDGE] ( A ) ( B ) ( C ) ( D ) ( E ) /=====BRIDGE=====/ ( ) ( ) ( ) ( ) ( ) [END-EDGE]
[START-EDGE] ( C ) ( D ) ( E ) ( ) ( ) /=====BRIDGE=====/ ( ) ( ) ( ) ( B ) ( A ) [END-EDGE]
[START-EDGE] ( C ) ( D ) ( E ) ( A ) ( ) /=====BRIDGE=====/ ( ) ( ) ( ) ( ) ( B ) [END-EDGE]
[START-EDGE] ( D ) ( E ) ( ) ( ) ( ) /=====BRIDGE=====/ ( ) ( ) ( ) ( B ) ( C ) ( A ) [END-EDGE]
[START-EDGE] ( D ) ( E ) ( A ) ( ) ( ) /=====BRIDGE=====/ ( ) ( ) ( ) ( B ) ( C ) ( B ) [END-EDGE]
[START-EDGE] ( A ) ( ) ( ) ( ) ( ) /=====BRIDGE=====/ ( ) ( ) ( B ) ( C ) ( D ) ( E ) [END-EDGE]
[START-EDGE] ( A ) ( B ) ( ) ( ) ( ) /=====BRIDGE=====/ ( ) ( ) ( C ) ( D ) ( E ) [END-EDGE]
[START-EDGE] ( ) ( ) ( ) ( ) ( ) /=====BRIDGE=====/ ( C ) ( D ) ( E ) ( B ) ( A ) [END-EDGE]
Solution has been found! Total Elapsed Time: 29
```

(Brute Force kullanılarak gerçekleştirilen örnek bir başarılı çözüm. 29 dakikada karşı tarafa geçiş sağlama başarılı olmuş.)

```
For Solution Number 1 -> Time until finding the solution : 3.8545989990234375 seconds.
For Solution Number 2 -> Time until finding the solution : 10.043643951416016 seconds.
For Solution Number 3 -> Time until finding the solution : 4.50875186920166 seconds.
For Solution Number 4 -> Time until finding the solution : 2.2330119609832764 seconds.
For Solution Number 5 -> Time until finding the solution : 5.6727070808410645 seconds.
For Solution Number 6 -> Time until finding the solution : 4.61078405380249 seconds.
For Solution Number 7 -> Time until finding the solution : 7.332798957824707 seconds.
For Solution Number 8 -> Time until finding the solution : 5.864474058151245 seconds.
For Solution Number 9 -> Time until finding the solution : 4.6454150676727295 seconds.
For Solution Number 10 -> Time until finding the solution : 2.9889309406280518 seconds.
For Solution Number 11 -> Time until finding the solution : 7.498455762863159 seconds.
For Solution Number 12 -> Time until finding the solution : 15.900845050811768 seconds.
For Solution Number 13 -> Time until finding the solution : 10.498047113418579 seconds.
For Solution Number 14 -> Time until finding the solution : 20.969747066497803 seconds.
For Solution Number 15 -> Time until finding the solution : 7.096193790435791 seconds.
For Solution Number 16 -> Time until finding the solution : 7.113417863845825 seconds.
For Solution Number 17 -> Time until finding the solution : 26.841169118881226 seconds.
For Solution Number 18 -> Time until finding the solution : 19.674826860427856 seconds.
For Solution Number 19 -> Time until finding the solution : 19.117125749588013 seconds.
For Solution Number 20 -> Time until finding the solution : 0.4154329299926758 seconds.
For Solution Number 21 -> Time until finding the solution : 2.868640899658203 seconds.
For Solution Number 22 -> Time until finding the solution : 16.47625184059143 seconds.
For Solution Number 23 -> Time until finding the solution : 1.8082480430603027 seconds.
For Solution Number 24 -> Time until finding the solution : 11.061588048934937 seconds.
For Solution Number 25 -> Time until finding the solution : 1.918982982635498 seconds.
```

(25 hesaplama turunun her birindeki çözümü buluş süresi kullanıcıya gösterilir. 25 hesaplama sırasında karşılaşılan en kısa çözüm süresi 0.41 saniye iken, en uzun çözüm süresi 26.84 saniye olmuş.)

AVERAGE TIME TO FIND THE SOLUTION: 8.84056360244751 seconds.

(Buna ek olarak bir çözümün bulunması için harcanan ortalama süre ise 8.84 saniye olmuş.)

```
For Solution Number 1 -> Number of epochs to find the solution : 216
For Solution Number 2 -> Number of epochs to find the solution : 603
For Solution Number 3 -> Number of epochs to find the solution : 258
For Solution Number 4 -> Number of epochs to find the solution : 131
For Solution Number 5 -> Number of epochs to find the solution : 346
For Solution Number 6 -> Number of epochs to find the solution : 278
For Solution Number 7 -> Number of epochs to find the solution : 433
For Solution Number 8 -> Number of epochs to find the solution : 352
For Solution Number 9 -> Number of epochs to find the solution : 280
For Solution Number 10 -> Number of epochs to find the solution : 169
For Solution Number 11 -> Number of epochs to find the solution : 451
For Solution Number 12 -> Number of epochs to find the solution : 949
For Solution Number 13 -> Number of epochs to find the solution : 602
For Solution Number 14 -> Number of epochs to find the solution : 1236
For Solution Number 15 -> Number of epochs to find the solution : 417
For Solution Number 16 -> Number of epochs to find the solution : 428
For Solution Number 17 -> Number of epochs to find the solution : 1559
For Solution Number 18 -> Number of epochs to find the solution : 1128
For Solution Number 19 -> Number of epochs to find the solution : 1106
For Solution Number 20 -> Number of epochs to find the solution : 28
For Solution Number 21 -> Number of epochs to find the solution : 155
For Solution Number 22 -> Number of epochs to find the solution : 938
For Solution Number 23 -> Number of epochs to find the solution : 111
For Solution Number 24 -> Number of epochs to find the solution : 641
For Solution Number 25 -> Number of epochs to find the solution : 110
```

(Brute Force kullanılarak yapılan 25 turluk çözüm sürecinde çözüme ulaşana kadar geçen rasgele çözüm turlarının her biri kullanıcıya gösterilir. En kısa çözüm turu 28 olmuşken, en fazla çözüm turu olarak 1559 diyebiliriz.)

AVERAGE EPOCH AMOUNT TO FIND THE SOLUTION: 517.0

(Çözümü bulana dek geçen ortalama epoch sayısı ortalaması 517 olmuş.)

2.4. Çözüm Başarısı

A* Algoritması kullanılarak problemin çözümünde köprüyü geçiş sürecini 32 dakikaya kadar indirmeyi başardım. Ancak 30 dakika olan toplam meşale yanış süresinin altına inmeye başarı sağlayamadı.

Buna çözüm üreten Brute Force yöntemi sayesinde 30 dakikanın altında çözümlere erişmeyi başardım. Aşağıdaki tabloda çözüm değerlerine ulaşmak için harcanan süreler ve tur sayılarına ulaşabilirsiniz.

ÇÖZÜM ARAŞTIRMA TURU	ÇÖZÜM SÜRESİ
1	~3.855 saniye
2	~10.044 saniye
3	~4.509 saniye
4	~2.233 saniye
5	~5.673 saniye
6	~4.611 saniye
7	~7.333 saniye
8	~5.864 saniye
9	~4.646 saniye
10	~2.989 saniye
11	~7.498 saniye
12	~15.901 saniye
13	~10.498 saniye
14	~20.970 saniye
15	~7.096 saniye
16	~7.113 saniye
17	~26.841 saniye
18	~19.675 saniye
19	~19.117 saniye
20	~0.416 saniye
21	~2.869 saniye
22	~16.476 saniye
23	~1.808 saniye
24	~11.062 saniye
25	~1.919 saniye

Ortalama	~8.841 saniye
----------	---------------

ÇÖZÜM ARAŞTIRMA TURU	ÇÖZÜM TUR SAYISI
1	216
2	603
3	258
4	131
5	346
6	278
7	433
8	352
9	280
10	169
11	451
12	949
13	602
14	1236
15	417
16	428
17	1559
18	1128
19	1106
20	28
21	155
22	938
23	111
24	641
25	110
Ortalama	517

A* Yöntemiyle bulunan çözüm sırası :

- A ve B karşıya geçer. Geçen süre : 3 saniye . Toplam Süre : 3 saniye
- A geri döner. Geçen süre : 1 saniye . Toplam süre : 4 saniye
- A ve C karşıya geçer. Geçen süre : 6 saniye. Toplam süre : 10 saniye
- A geri döner. Geçen süre : 1 saniye. Toplam süre : 11 saniye
- A ve D karşıya geçer. Geçen süre : 8 saniye. Toplam süre : 19 saniye
- A geri döner. Geçen süre : 1 saniye. Toplam süre : 20 saniye
- A ve E karşıya geçer. Geçen süre : 12 saniye . **Toplam süre : 32 saniye**

Brute Force Yöntemiyle bulunan çözüm sırası :

- A ve B karşıya geçer. Geçen süre : 3 saniye. Toplam süre : 3 saniye.
- A geri döner. Geçen süre : 1 saniye. Toplam süre : 4 saniye.
- A ve C karşıya geçer. Geçen süre : 6 saniye. Toplam süre : 10 saniye.
- A geri döner. Geçen süre : 1 saniye. Toplam süre : 11 saniye
- D ve E karşıya geçer. Geçen süre : 12 saniye. Toplam süre : 23 saniye
- B geri döner. Geçen süre : 3 saniye. Toplam süre : 26 saniye
- A ve B karşıya geçer. Geçen süre : 3 saniye. **Toplam süre : 29 saniye**

3. Genetik Algoritmalarla Gezgin Satıcı Probleminin Çözümü (**Travelling Salesman Problem / TSP**) (3. Alternatif)

3.1 Gezgin Satıcı Problemi Tanımı

Bir seyyar satıcı var. Bu seyyar satıcı mallarını toplamda N adet şehirde dağıtmak istiyor. Bu dağıtım, tüm bu şehirlere maksimum bir kez uğramak koşuluyla mümkün olan en kısa yolları takip ederek gerçekleştirmek istiyor.

Problem Kompleksite Kümesi : NP-Complete

Kullandığım Çözüm Yöntemi : Genetik Algoritma

Kullandığım Dil : Python

NOT: Programımda problem tanımı üzerine belli modifikasyonlarda bulundum.

Tanımladığım problemin koşulları aşağıdaki şekilde:

- Bir lojistik şirketi Türkiye'nin 81ehrine Covid-19 enfeksiyonuna yardımcı olmak adına N95 maske dağıtımları gerçekleştirmek istemektedir.*
- Bu maskeleri iletmek için optimize ve uygun bir rota belirlemek istemektedirler.*
- Depolar bu maskeleri acilen istemektedir dolayısı ile olabilecek en iyi çözüme erişmek için yeteri kadar zaman yoktur.*
- Belirtilen süre içerisinde şirketin optimal bir çözüm bulması gerekmektedir.*

-Şehir (Node) Sayısı: 81

-Maksimum süre: 5 dakika (5 dakika içerisinde 10 kez mevcut olan en iyi optimize çözüme erişilmeli / tur başına yaklaşık 30 saniye)

3.2 Programa Dair Özellikler, Sınıf Yapıları ve Gerekli Açıklamalar

Python dilini kullanarak gerekli genetik algoritmayı Object Oriented Programming ilkelerine uygun bir biçimde simüle etmek için çaba gösterdim. Programımda toplamda 8 adet modül bulunuyor. Bu modülleri aşağıdaki gibi kategorize edebiliriz.

Test modülleri : Test

Genetik Algoritma modülleri : Evolution, Mutation, MatingPool, Fitness, FitnessEvaluation, Population

Aktör modülleri : City

City sınıfı:

Bu sınıfta iki boyutlu bir uzay içerisinde yer alan şehirler, x düzlemi ve y düzlemi üzerindeki noktasal konumları ile tanımlanan objelerdir. City objesinin sahip olduğu metodlar:

distance () :

- parametreler: city(City)
- return : distance(double)

Bu metot bir şehrin, diğer bir şehir ile arasındaki mesafenin hesaplanması sağlayan metottur. Öklit uzunluğu prensibine bağlı olarak çalışır ve mesafeyi geri döndürür.

Population sınıfı:

Bu sınıf City objelerinin yanı şehirlerin yer aldığı bir uzay meydana getirilmesinde, bu şehirler arasında seyahat etmede kullanılabilecek çeşitli rotaların ve bu rotalardan oluşan rota popülasyonlarının meydana getirilmesinde görevli metodlardan oluşmaktadır.

create_cities() :

- parametreler : city_amount(int) , max_distance_xy(int)
- return : city_list(list)

Bu metot parametre olarak aldığı city_amount değişkeninde belirtildiği kadar şehri içeren bir uzay yaratır. Bu uzayın içerisinde max_distance_xy değişkeninde belirtilen maksimum değeri aşmayacak şekilde koordinat değerlerine sahip şehirler yerleştirir. En sonunda oluşturduğu şehirleri geri döndürür.

create_route() :

- parametreler : city_list(list)
- return : route

Bu metot parametre olarak şehirlerin listesini alır ve mevcut şehirler arasında gerçekleşmesi mümkün olası bir rasgele rota yaratır. Sonra bu rotayı döndürür.

initial_population():

- parametreler : pop_size(int), city_list(list)
- return : population(list)

Bu metot pop_size değişkeninde belirtilen miktarda farklı rota yaratır. Bu rotalar create_route fonksiyonu kullanılarak rasgele olarak meydana getirilmişlerdir. Son olarak oluşturulan rasgele özelliklere sahip popülasyon geri döndürülür.

Fitness sınıfı:

Bu sınıf bir rotanın uyumluluk değerini ifade etmek için yaratılmış sınıftır. Bu sınıfın içerisinde aşağıdaki metodlar yer alır.

`route_distance() :`

-parametreler : void

-return : distance(double)

Bu metot bir rotanın toplam uzunluğunun, başka bir deyişle maliyetinin hesaplanması için kullanılan fonksiyondur. Hesaplandıktan sonra sonucu kendi içsel değişkenin kaydeder ve ardından döndürür.

`route_fitness() :`

-parametreler : void

-return : fitness(double)

Bu metot rotanın uyumluluğunu basit bir şekilde 1/toplam yol uzunluğu formülünü kullanarak hesaplamaktadır. Bulduğu değeri sınıfın kendi içsel değişkenlerine kaydeder ve geri döndürür.

Fitness Evaluation Sınıfı:

Bu sınıfın içerisinde popülasyona ait uyumluluk değerlerinin hesaplanması görev alan metot yer almaktadır.

`rank_routes() :`

-parametreler : population(list)

-return : population(list)

Bu metot popülasyona ait rotaların/bireylerin her birinin uyumluluk değerlerinin hesapmasını ve ardından söz konusu popülasyonun bu uyumluluk değerlerine bağlı olarak sıralanmasını sağlar. Bu sayede popülasyon içerisindeki rotaları/bireyleri en uyumlu olandan en uyumsuz olana doğru sıralamış oluruz. Bu durum seçim aşamasını kolaylaştırır. En son sıralanmış popülasyon geri döndürülür.

MatingPool sınıfı:

Bu sınıfın içerisinde popülasyon içerisindeki elit bireylerin seçilmesi, bu elit bireyler kullanılarak uygun bir çifteşme havuzu oluşturulması, seçilmiş ebeveyn bireylerin çiftleştirilmesi ve tüm popülasyona yönelik bir yavru havuzu oluşturulmasını sağlayan metotlar yer almaktadır.

selection () :

-parametreler : pop_ranked(list) , elite_pop_size(int)
-return : selection_results(list)

Bu metot sıralanmış popülasyon içerisinde belirtilen elit popülasyon limiti dolana kadar seçkin bireyleri seçmekle görevlidir. En son seçilmiş bireyler geri döndürülür.

mating_pool():

-parametreler : population(list), selection_results(list)
-return : matingpool(list)

Bu metot popülasyonun içerisinde yer alan kişilerden sadece selection_results metodu kullanılarak seçkin olduğu belirlenmiş olanların alınarak bir çifteşme havuzu oluşturulmasını sağlar. En son bu çifteşme havuzunu geri döndürür.

breed () :

-parametreler : parent1, parent2
-return : child

Bu metot ebeveyn bireylerden gelen özelliklerini kullanarak yavru bireylerin oluşturulmasını sağlayan metottur. Oluşturulan yavru bireyi geri döndürür.

breed_population():

-parametreler : matingpool(list), elite_pop_size(int)
-return : children (list)

Bu metot dahilinde popülasyona ait elit birey sayısı ve çifteşme havuzu dahilinde yeni popülasyonu meydana getirecek olan yavru bireyler oluşturulur. En son yavru bireyler geri döndürülür.

Mutation sınıfı:

Bu sınıfı bireylerin ve popülasyonların rasgele mutasyonlar ile değişimelerini sağlama amacı güden metodlar bulunmaktadır.

`mutate() :`

-parametreler : individual, mutation_rate(int)

-return : individual

Bu metot tekil bir bireyin rota güzergahı üzerindeki rasgele iki noktanın yer değiştirmesini sağlama yoluyla rota kromozomu üzerinde mutasyon sağlamayı amaçlar. Bu mutasyon olasılıksaldır ve mutation_rate değişkeninde belirtilen oranda gerçekleşir. En sonunda mutasyona uğramış veya uğramamış birey geri döndürülür.

`mutate_population():`

-parametreler : population(list), mutation_rate(int)

-return : mutated_population(list)

Bu metot popülasyonların mutasyona uğrama durumunu denetlemek ve gerçekleştirmek için dizayn edilmiş bir metottur. Sırasıyla popülasyonun içerisindeki tüm bireyler için mutate fonksiyonu çağrılır ve bu mutasyonun riskini mutasyon şansı değişkeni denetler.

Evolution Sınıfı:

Bu sınıfı genetik algoritmanın faaliyete geçirilmesini sağlayan temel fonksiyonların çağrılmasını ve jenerasyonların ilerletilmesini sağlayan kilit(wrapper) fonksiyonlar yer almaktadır.

`next_generation():`

-parametreler : current_gen(list), elite_pop_size(int), mutation_rate(int)

-return : next_generation(list)

Bu metot sınırları dahilinde popülasyonu ilerletmek ile ilgili uyumluluk hesaplama, seçim, üreme havuzu oluşturma, çiftleştirme ve mutasyona uğratma gibi gerekli metodların yapılması yoluyla yeni popülasyonun meydana getirilmesini sağlar. Sonuç olarak yeni jenerasyonu döndürür.

`genetic_algorithm():`

-parametreler : pop_size(int), elite_size(int), mutation_rate(int), generations(int), graph_pieces(int), city_amount(int), max_distance_xy(int)

-return : best_route, progress

Bu metot genetik algoritmanın çalışmasını ve genetik algoritmanın ilerleme süreciyle ilgili kullanıcıya bilgi sağlayan görselleştirmelerin gerçekleştirilmesini sağlayan fonksiyondur. Fonksiyon içine dahil ettiği parametrelere bağlı olarak genetik algoritmayı işlerlige sokar ve sonucunda elde ettiği en iyi rotayı geri döndürür.

Test sınıfı:

Bu sınıf programın işlerliğinin test edilmesi için yaratılmış olup, öncelikli olarak genetik algoritmanın çalışması için gerekli olan hiper parametrelerin tanımlarını içermekte, ardından genetik algoritmanın toplamda 10 tur boyunca çalıştırılmasını sağlamaktadır. Bu çalışma turlarının her birinin ardından sonuçları kaydetmekte ve bu sonuçların her birini ve ilgili diğer metrikleri kullanıcıya iletmektedir.

Programda görselleştirme kütüphanesi olarak **matplotlib.pyplot** kullanılmıştır.

3.3 Kod Görüntüleri ve Program Çıktıları

Kod Görüntüleri:

```
import Evolution
import Population
import time
import matplotlib.pyplot as plt

#Define the hyperparameters that will be used in genetic algorithm
pop_size = 100
elite_pop_size = 20
mutation_rate = 0.01
generations = 100
city_amount = 81
max_distance_xy = 2000
graph_pieces = 2

#Maximum time to find solutions (minutes)
total_time = 5
total_turns = 10

time_share = (total_time*60)/total_turns

#Define a common city list (problem) for each turn
city_list = Population.create_cities(city_amount, max_distance_xy)
```

```

#Start the genetic algorithm
history = []
for i in range(0, 10):
    print("TURN NUMBER : ", (i+1))
    start = time.time()

#This code works depending on the time limit and turn amount. It gives each turn an equal time share.
bestRoute, progress = Evolution.genetic_algorithm(pop_size,
                                                elite_pop_size,
                                                mutation_rate,
                                                generations,
                                                graph_pieces,
                                                city_amount,
                                                max_distance_xy,
                                                city_list,
                                                time_share)

#This code ignores the time limit and trains the genetic algorithm based on the number of generations
"""
bestRoute, progress = Evolution.genetic_algorithm(pop_size,
                                                elite_pop_size,
                                                mutation_rate,
                                                generations,
                                                graph_pieces,
                                                city_amount,
                                                max_distance_xy,
                                                city_list)
"""

#Time Share
#This code ignores the time limit and trains the genetic algorithm based on the number of generations
"""
bestRoute, progress = Evolution.genetic_algorithm(pop_size,
                                                elite_pop_size,
                                                mutation_rate,
                                                generations,
                                                graph_pieces,
                                                city_amount,
                                                max_distance_xy,
                                                city_list)
"""

end = time.time()
history.append(progress)

#print the best route for the final population
print("Best Route (x1, y1) -> (x2, y2) -> ... -> (xn, yn) : ")
print(bestRoute)

#print the total number of generations
print("Total Number of Generations : ", generations)

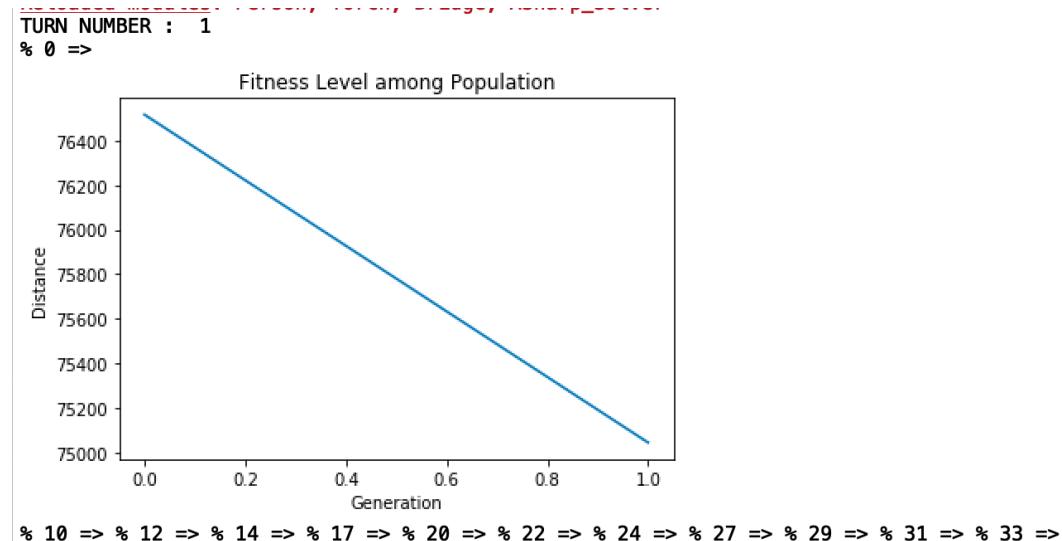
#print the total time of the calculation
elapsed_time = (end-start)
print("Total Time Elapsed for Turn : ", int(elapsed_time*1000) , " milliseconds.")

print("_____")

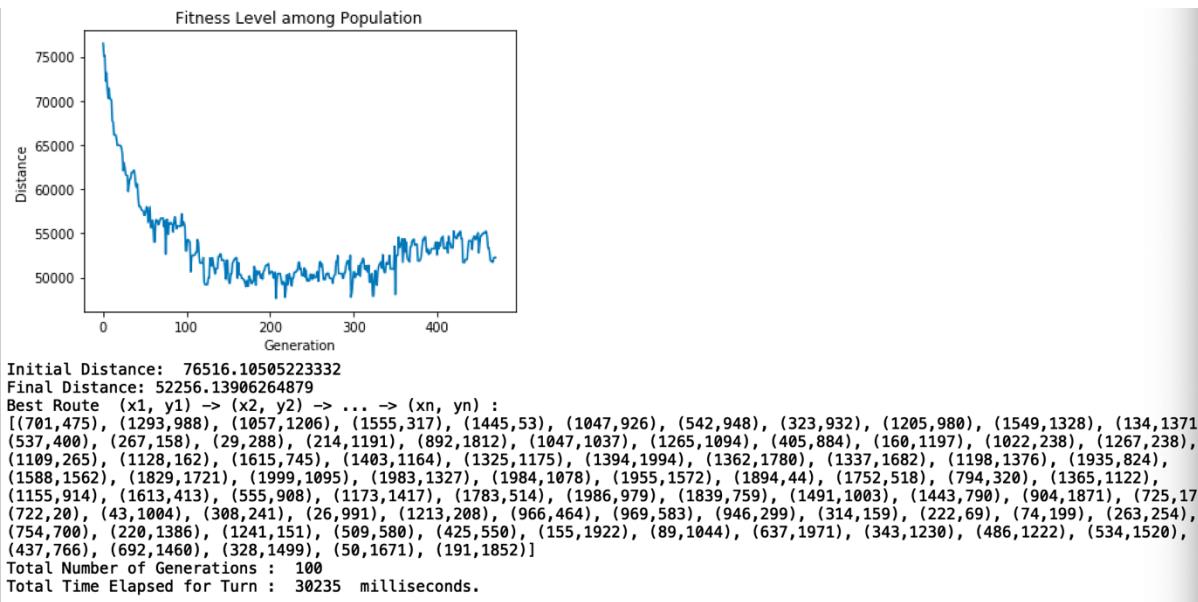
```

```
plt.title("Fitness Level among All Turns")
plt.ylabel('Distance')
plt.xlabel('Generation')
for i in range(0, len(history)):
    plt.plot(history[i])
plt.show()
```

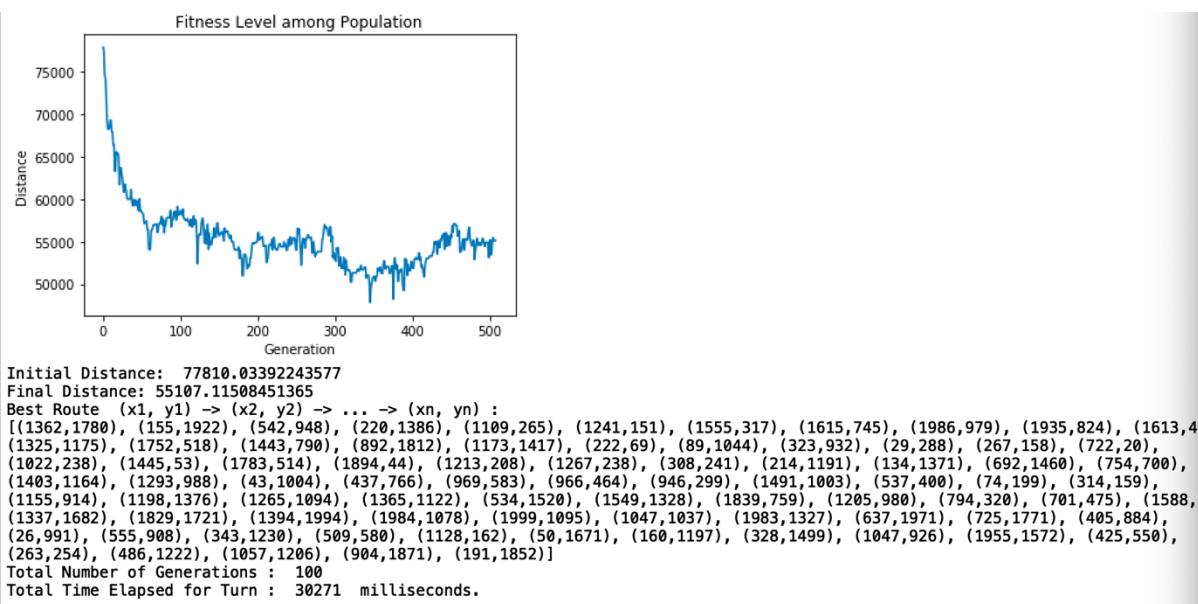
Program Çıktıları:



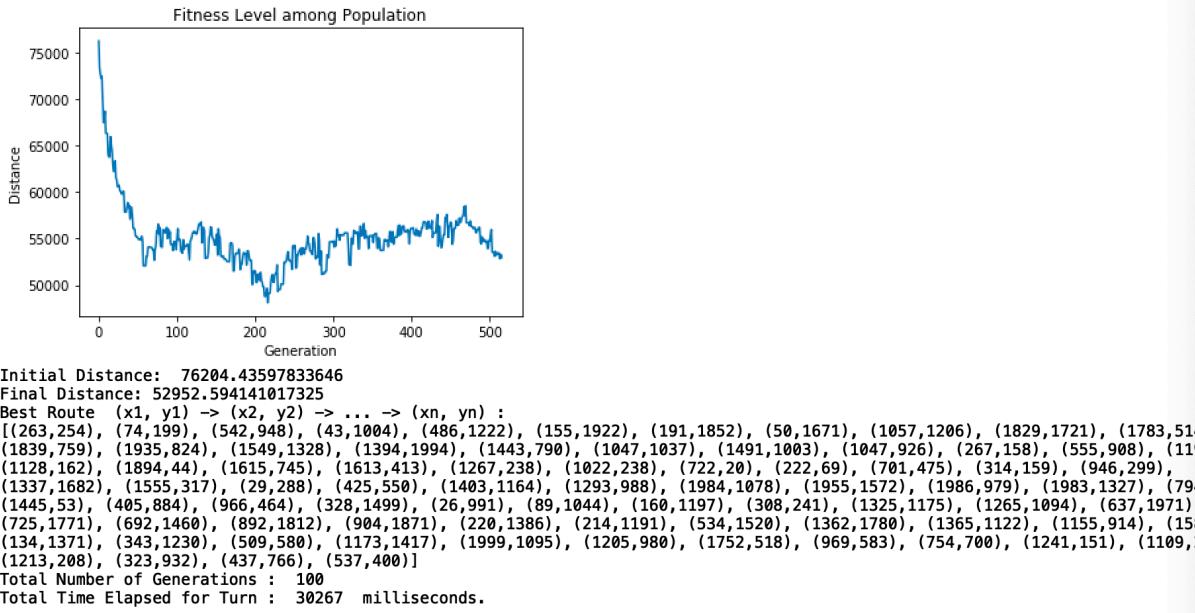
(Genetik algoritmanın hesaplama aşamasından bir görüntü)



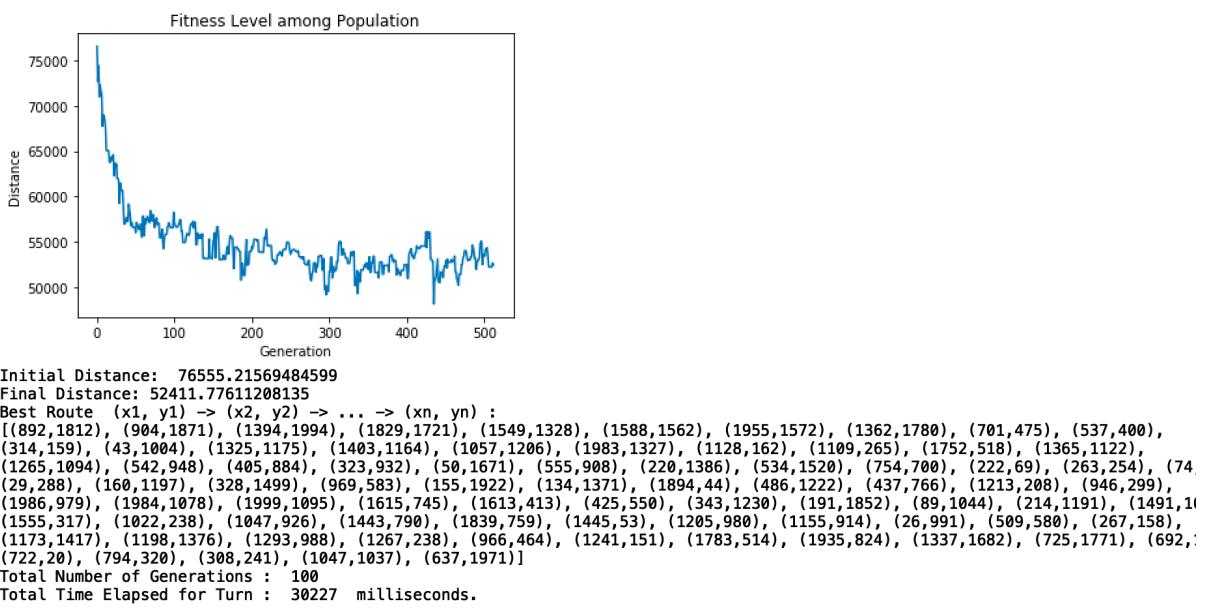
(Bir tur sonucunda bulunan optimize bir rotaya ait görüntü, bu rotaya erişebilmek için geçen süre ve jenerasyon sayısı)



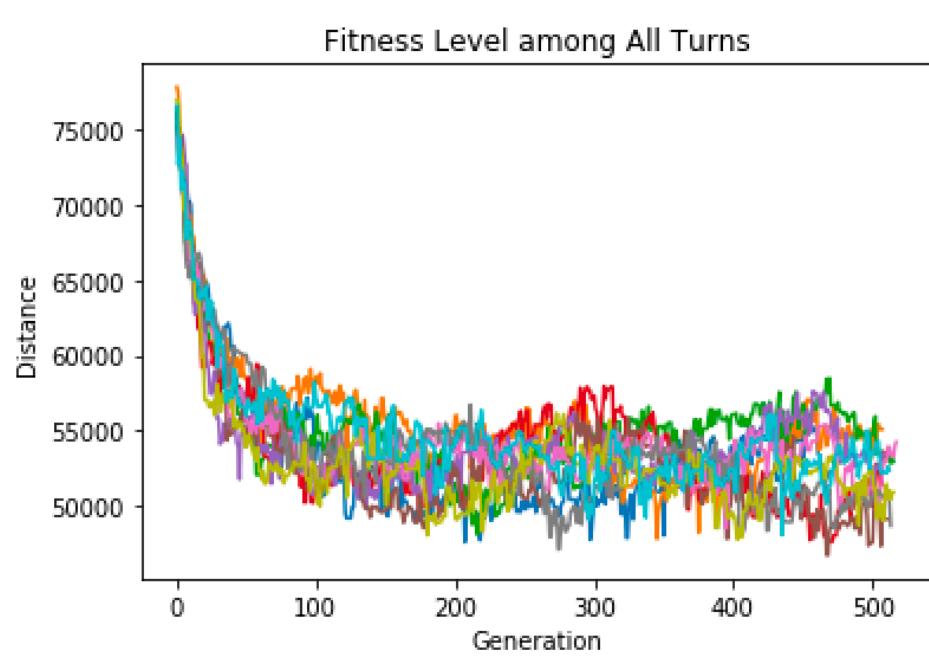
(Başka bir tur hesaplamaya ait sonuçlar)



(Başka bir tur hesaplamaya ait sonuçlar)



(Başka bir tur hesaplamaya ait sonuçlar)



(10 hesaplama turu sonrasında optimize edilen seyahat rotalarının hepsinin bir grafik üzerindeki görüntüsü. Problemde arayışında olunan çözüme uygun şekilde 80.000 civarında ifade edilen mesafelerden 45.000-55.000 arası uzaklıklara indirgenen bir optimizasyon sağlandığını gözlemleyebiliyoruz.)

3.3 Etkin Rota Örnekleri (Sırasıyla şehir koordinatları)

```
[(892,1812), (904,1871), (1394,1994), (1829,1721), (1549,1328), (1588,1562), (1955,1572), (1362,1780), (701,475), (537,400), (314,159), (43,1004), (1325,1175), (1403,1164), (1057,1206), (1983,1327), (1128,162), (1109,265), (1752,518), (1365,1122), (1265,1094), (542,948), (405,884), (323,932), (50,1671), (555,908), (220,1386), (534,1520), (754,700), (222,69), (263,254), (74,199), (29,288), (160,1197), (328,1499), (969,583), (155,1922), (134,1371), (1894,44), (486,1222), (437,766), (1213,208), (946,299), (1986,979), (1984,1078), (1999,1095), (1615,745), (1613,413), (425,550), (343,1230), (191,1852), (89,1044), (214,1191), (1491,1003), (1555,317), (1022,238), (1047,926), (1443,790), (1839,759), (1445,53), (1205,980), (1155,914), (26,991), (509,580), (267,158), (1173,1417), (1198,1376), (1293,988), (1267,238), (966,464), (1241,151), (1783,514), (1935,824), (1337,1682), (725,1771), (692,1460), (722,20), (794,320), (308,241), (1047,1037), (637,1971)]
```

(toplam mesafe : ~52411)

```
[(1752,518), (1613,413), (1555,317), (1109,265), (701,475), (794,320), (969,583), (1935,824), (1986,979), (1984,1078), (1783,514), (1241,151), (1839,759), (29,288), (74,199), (314,159), (323,932), (405,884), (1337,1682), (1293,988), (43,1004), (267,158), (222,69), (26,991), (160,1197), (437,766), (966,464), (1267,238), (1022,238), (946,299), (1894,44), (214,1191), (425,550), (555,908), (220,1386), (754,700), (725,1771), (542,948), (1615,745), (1205,980), (1443,790), (1588,1562), (1394,1994), (692,1460), (1057,1206), (1365,1122), (1325,1175), (1047,926), (263,254), (308,241), (486,1222), (637,1971), (722,20), (537,400), (1213,208), (1445,53), (1128,162), (1265,1094), (1955,1572), (1999,1095), (343,1230), (191,1852), (50,1671), (509,580), (134,1371), (89,1044), (1198,1376), (1983,1327), (1549,1328), (1829,1721), (1362,1780), (892,1812), (155,1922), (328,1499), (904,1871), (1173,1417), (534,1520), (1403,1164), (1047,1037), (1155,914), (1491,1003)]
```

(toplam mesafe : ~50921)

```
[(892,1812), (904,1871), (1394,1994), (1829,1721), (1549,1328), (1588,1562), (1955,1572), (1362,1780), (701,475), (537,400), (314,159), (43,1004), (1325,1175), (1403,1164), (1057,1206), (1983,1327), (1128,162), (1109,265), (1752,518), (1365,1122), (1265,1094), (542,948), (405,884), (323,932), (50,1671), (555,908), (220,1386), (534,1520), (754,700), (222,69), (263,254), (74,199), (29,288), (160,1197), (328,1499), (969,583), (155,1922), (134,1371), (1894,44), (486,1222), (437,766), (1213,208), (946,299), (1986,979), (1984,1078), (1999,1095), (1615,745), (1613,413), (425,550), (343,1230), (191,1852), (89,1044), (214,1191), (1491,1003), (1555,317), (1022,238), (1047,926), (1443,790), (1839,759), (1445,53), (1205,980), (1155,914), (26,991), (509,580), (267,158), (1173,1417), (1198,1376), (1293,988), (1267,238), (966,464), (1241,151), (1783,514), (1935,824), (1337,1682), (725,1771), (692,1460), (722,20), (794,320), (308,241), (1047,1037), (637,1971)]
```

(toplam mesafe : ~48706)

[(1549,1328), (1935,824), (946,299), (323,932), (50,1671), (191,1852), (1047,1037), (220,1386), (134,1371), (509,580), (892,1812), (1173,1417), (1894,44), (1555,317), (1752,518), (1839,759), (1783,514), (1986,979), (1155,914), (1443,790), (1293,988), (754,700), (214,1191), (405,884), (555,908), (437,766), (486,1222), (328,1499), (1362,1780), (722,20), (308,241), (267,158), (89,1044), (794,320), (692,1460), (969,583), (263,254), (1047,926), (1267,238), (1241,151), (314,159), (74,199), (425,550), (534,1520), (904,1871), (1198,1376), (637,1971), (725,1771), (160,1197), (537,400), (1445,53), (1128,162), (1022,238), (1613,413), (1205,980), (1394,1994), (155,1922), (1057,1206), (1109,265), (26,991), (43,1004), (343,1230), (1984,1078), (1955,1572), (1337,1682), (1403,1003), (1325,1175), (1337,1682), (1265,1094), (1198,1376), (1984,1078), (1935,824), (1986,979), (1555,317), (1613,413), (1205,980), (1155,914), (1403,1164)]

(toplasm mesafe : ~54259)

[(637,1971), (904,1871), (1549,1328), (1173,1417), (155,1922), (191,1852), (1394,1994), (1999,1095), (1839,759), (1443,790), (1615,745), (1752,518), (1047,1037), (1293,988), (1983,1327), (1445,53), (1047,926), (542,948), (328,1499), (89,1044), (43,1004), (537,400), (486,1222), (701,475), (1213,208), (1894,44), (754,700), (314,159), (308,241), (425,550), (966,464), (1109,265), (1128,162), (692,1460), (969,583), (1055,1572), (725,1771), (1362,1780), (1588,1562), (134,1371), (534,1520), (405,884), (794,320), (1241,151), (722,20), (343,1230), (214,1191), (160,1197), (1057,1206), (1829,1721), (267,158), (946,299), (1267,238), (1022,238), (1613,413), (1205,980), (1155,914), (1403,1164), (1047,1037), (1337,1682), (89,1044), (534,1520), (328,1499), (343,1230), (1128,162), (1022,238), (1445,53), (1213,208), (267,158), (314,159), (74,199), (425,550), (1047,926), (542,948), (1783,514), (1365,1122), (1205,980), (1986,979), (1549,1328), (1588,1562), (1955,1572), (1983,1327), (308,241), (29,288), (437,766), (509,580), (323,932), (50,1671), (26,991), (220,1386), (555,908), (969,583), (1325,1175), (1337,1682), (1265,1094), (1198,1376), (1984,1078), (1935,824), (1986,979), (1555,317), (1613,413), (1205,980), (1155,914), (1403,1164)]

(toplasm mesafe : ~49142)

[(1265,1094), (1109,265), (1267,238), (134,1371), (155,1922), (50,1671), (43,1004), (160,1197), (222,69), (1241,151), (1555,317), (1894,44), (1752,518), (946,299), (722,20), (904,1871), (1198,1376), (26,991), (486,1222), (1935,824), (1491,1003), (214,1191), (754,700), (969,583), (509,580), (263,254), (701,475), (1213,208), (1894,44), (754,700), (314,159), (308,241), (425,550), (966,464), (1109,265), (1128,162), (692,1460), (969,583), (1055,1572), (725,1771), (1362,1780), (1588,1562), (134,1371), (534,1520), (405,884), (794,320), (1241,151), (722,20), (343,1230), (214,1191), (160,1197), (1057,1206), (1829,1721), (267,158), (946,299), (1267,238), (1022,238), (1613,413), (1205,980), (1155,914), (1403,1164), (1047,1037), (1337,1682), (89,1044), (534,1520), (328,1499), (343,1230), (1128,162), (1022,238), (1445,53), (1213,208), (267,158), (314,159), (74,199), (425,550), (1047,926), (542,948), (1783,514), (1365,1122), (1205,980), (1986,979), (1549,1328), (1588,1562), (1955,1572), (1983,1327), (308,241), (29,288), (437,766), (509,580), (323,932), (50,1671), (26,991), (220,1386), (1325,1175), (1337,1682), (1265,1094), (1198,1376), (1984,1078), (1935,824), (1986,979), (1555,317), (1613,413), (1205,980), (1155,914), (1403,1164)]

(toplasm mesafe : ~53433)

[(555,908), (222,69), (89,1044), (1613,413), (1752,518), (1894,44), (1128,162), (1213,208), (1615,745), (1445,53), (263,254), (267,158), (74,199), (29,288), (405,884), (966,464), (1109,265), (1128,162), (692,1460), (969,583), (1055,1572), (725,1771), (1362,1780), (1588,1562), (134,1371), (534,1520), (405,884), (794,320), (1241,151), (722,20), (343,1230), (214,1191), (160,1197), (1057,1206), (1829,1721), (267,158), (946,299), (1267,238), (1022,238), (1613,413), (1205,980), (1155,914), (1403,1164), (1047,1037), (1337,1682), (89,1044), (534,1520), (328,1499), (343,1230), (1128,162), (1022,238), (1445,53), (1213,208), (267,158), (314,159), (74,199), (425,550), (1047,926), (542,948), (1783,514), (1365,1122), (1205,980), (1986,979), (1549,1328), (1588,1562), (1955,1572), (1983,1327), (308,241), (29,288), (437,766), (509,580), (323,932), (50,1671), (26,991)]

(toplasm mesafe : ~53584)

[(263,254), (74,199), (542,948), (43,1004), (486,1222), (155,1922), (191,1852), (50,1671), (1057,1206), (1829,1721), (1783,514), (1839,759), (1935,824), (1549,1328), (1394,1994), (1443,790), (1047,1037), (1491,1003), (1047,926), (267,158), (555,908), (1198,1376), (1128,162), (1894,44), (1615,745), (1613,413), (1267,238), (1022,238), (722,20), (701,475), (314,159), (946,299), (1337,1682), (1555,317), (29,288), (425,550), (1403,1164), (1293,988), (1984,1078), (1955,1572), (1986,979), (1983,1327), (794,320), (1445,53), (405,884), (966,464), (328,1499), (26,991), (89,1044), (160,1197), (308,241), (1325,1175), (1265,1094), (637,1971), (725,1771), (692,1460), (969,583), (1055,1572), (725,1771), (1362,1780), (1588,1562), (134,1371), (343,1230), (509,580), (1173,1417), (1999,1095), (1205,980), (1752,518), (969,583), (754,700), (1241,151), (1109,265), (1213,208), (323,932), (437,766), (537,400)]

(toplasm mesafe : ~52952)

[(1362,1780), (155,1922), (542,948), (220,1386), (1109,265), (1241,151), (1555,317), (1615,745), (1986,979), (1935,824), (1613,413), (1325,1175), (1752,518), (1443,790), (892,1812), (1173,1417), (222,69), (89,1044), (323,932), (29,288), (267,158), (722,20), (1022,238), (1445,53), (1894,44), (1047,1037), (1293,988), (1984,1078), (1955,1572), (1986,979), (1983,1327), (794,320), (1403,1164), (1293,988), (43,1004), (437,766), (969,583), (966,464), (946,299), (1491,1003), (537,400), (74,199), (314,159), (1155,914), (1198,1376), (1265,1094), (1365,1122), (534,1520), (1362,1780), (1588,1562), (1337,1682), (1829,1721), (1394,1994), (1984,1078), (1999,1095), (1047,1037), (1983,1327), (637,1971), (725,1771), (405,884), (26,991), (555,908), (343,1230), (509,580), (1128,162), (50,1671), (160,1197), (328,1499), (1047,926), (1955,1572), (425,550), (263,254), (486,1222), (1057,1206), (904,1871), (191,1852)]

(toplasm mesafe: ~51107)

[(701,475), (1293,988), (1057,1206), (1555,317), (1445,53), (1047,926), (542,948), (323,932), (1205,980), (1549,1328), (134,1371), (537,400), (267,158), (29,288), (214,1191), (892,1812), (1047,1037), (1265,1094), (405,884), (160,1197), (1022,238), (1267,238), (1109,265), (1128,162), (1615,745), (1403,1164), (1325,1175), (1394,1994), (1362,1780), (1337,1682), (1198,1376), (1935,824), (1588,1562), (1829,1721), (1999,1095), (1983,1327), (1984,1078), (1955,1572), (1894,44), (1752,518), (794,320), (1365,1122), (1155,914), (1613,413), (555,908), (1173,1417), (1783,514), (1986,979), (1839,759), (1491,1003), (1443,790), (984,1871), (725,1771), (1155,914), (1241,151), (509,580), (425,550), (155,1922), (89,1044), (637,1971), (343,1230), (486,1222), (534,1520), (437,766), (692,1460), (328,1499), (50,1671), (191,1852)]

(toplasm mesafe: ~52256)

4. Python ile Makine Öğrenmesi [Sınıflandırıcı]

4.1 Veri setinin ve Problemin Kısa Anlatımı

Veri seti *keras* kütüphanesine ait veri etlerinden seçilmiştir. Adı Fashion MNIST olan veri seti, içerisinde 10 farklı tipte giyim ürününe ait görüntüleri barındırmaktadır. Görüntüler siyah-beyaz olup boyutları 28'e 28'dir.

Eğitim Seti Veri Sayısı : **60.000 adet**

Test Seti Veri Sayısı : **10.000 adet**

Öznitelik Sayısı :

En : 28

Boy : 28

Derinlik : 1

olmak üzere toplamda $28 \times 28 = 784$ parametre barındırmaktadır.

Sınıf Sayısı : **10 adet**

Sınıflar :

T-Shirt

Trouser

Pullover

Dress

Coat

Sandal

Shirt

Sneaker

Bag

Ankle Boot

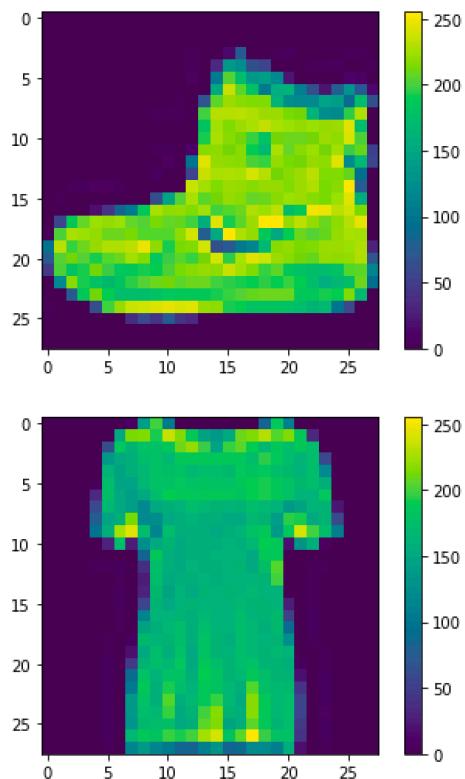
Sınıflandırma için Kullanılan Yöntemler:

Yöntem 1 : *MLP (Multilayer Perceptron)*

Yöntem 2 : *Random Forest*

TRAIN DATA SHAPE : (60000, 28, 28)
TRAIN LABELS LENGTH : 60000
TEST DATA SHAPE : (10000, 28, 28)
TEST LABELS LENGTH : 10000

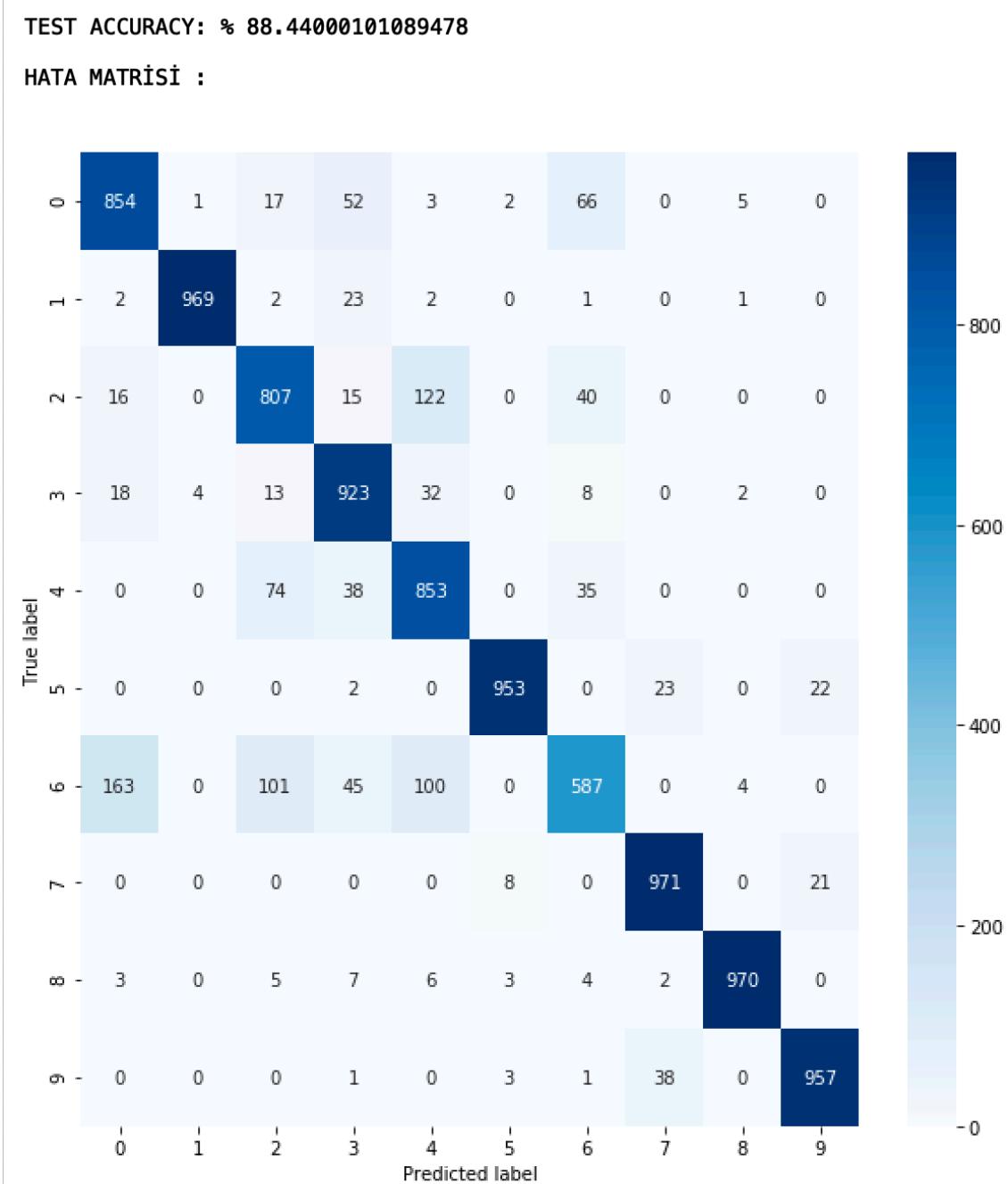
EXAMPLE DATA SAMPLES:



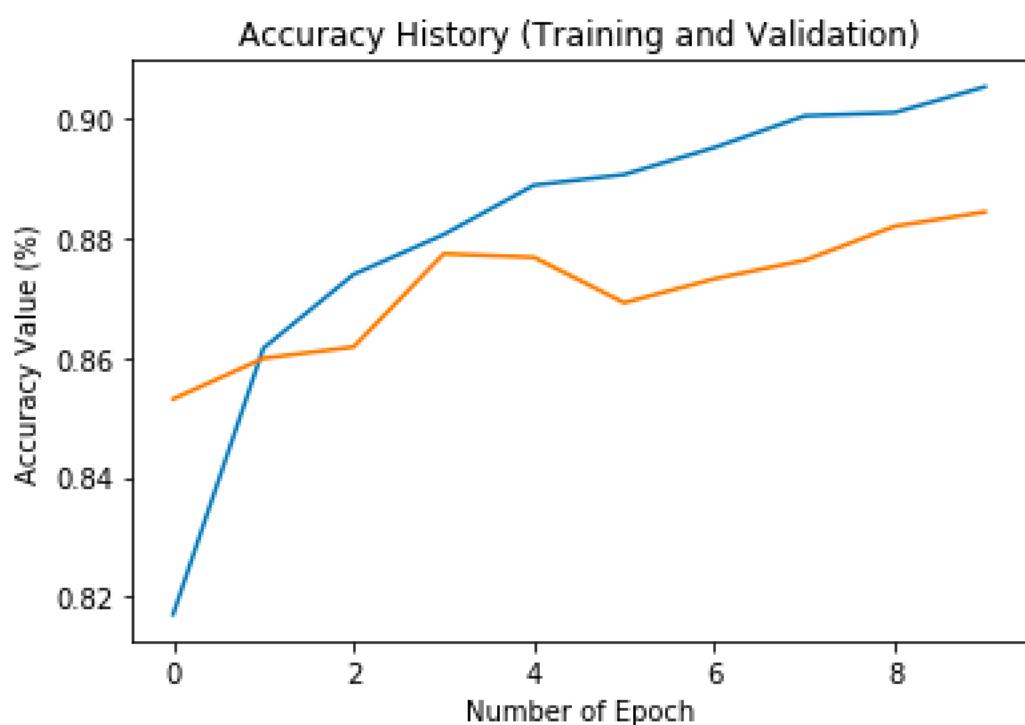
EXAMPLES FROM VARIOUS CLASSES :



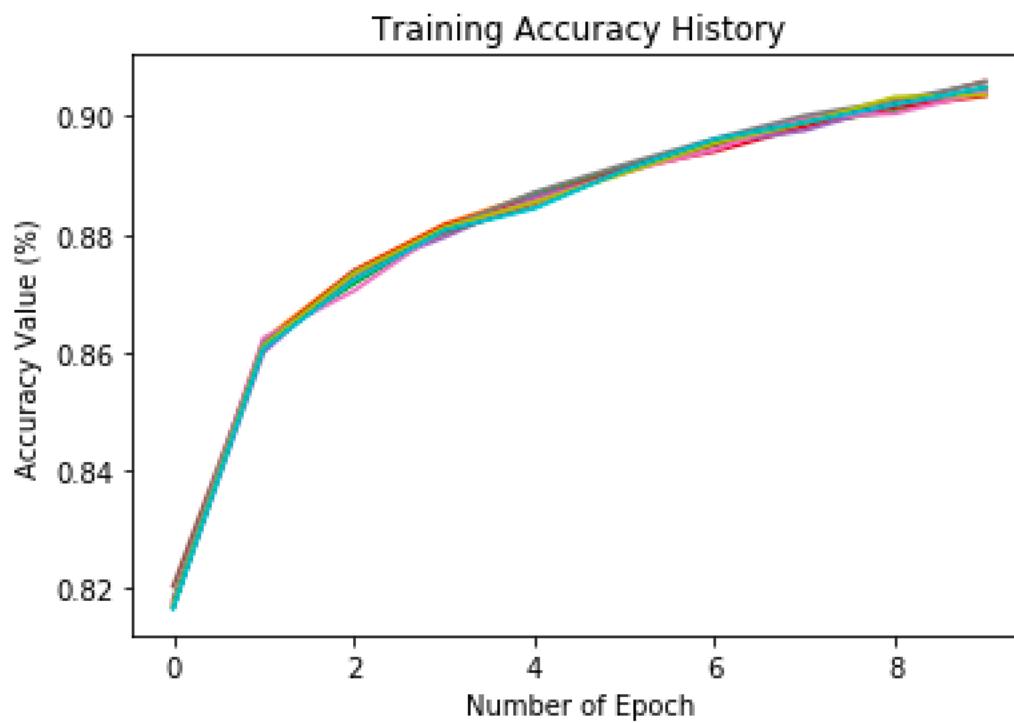
4.2 İki Farklı Sınıflandırıcı için Sonuçlar : Hata Matrisleri, Tablo



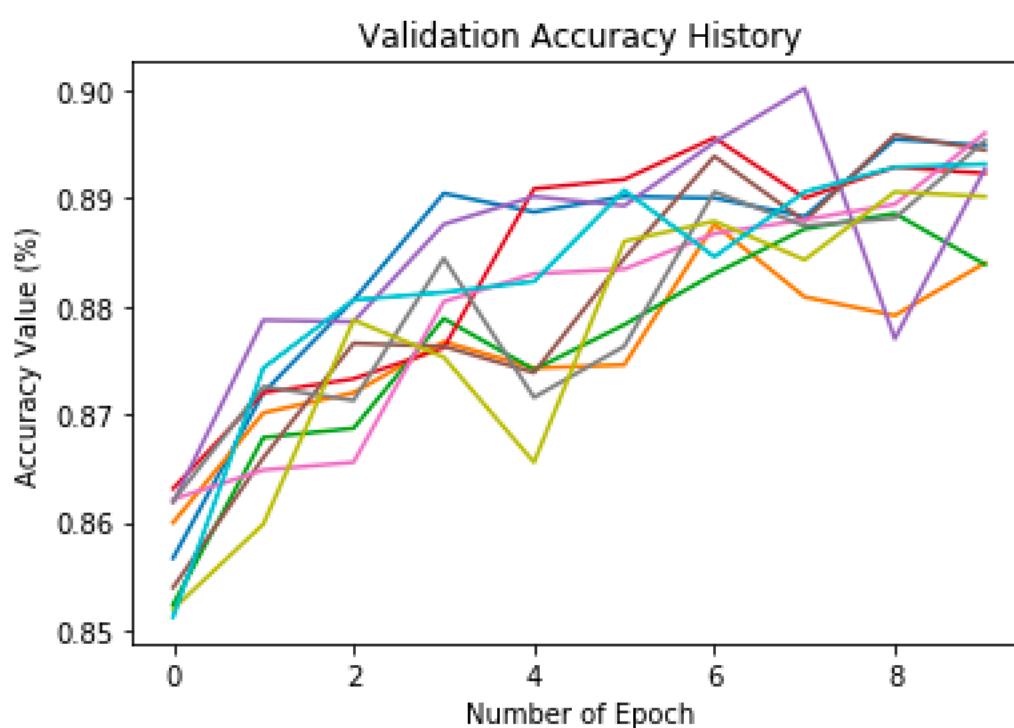
(MULTILAYER PERCEPTRON - ÇAPRAZ DOĞRULAMA OLmadan EĞiTİM SONUCU ELDE EDİLEN HATA MATRİSİ VE TEST SETİ DOĞRULUK ORANI)



**(MULTILAYER PERCEPTRON - ÇAPRAZ DOĞRULAMA OLMADAN EĞİTİM SIRASINDA
EĞİTİM VE TEST SETİNİN DOĞRULUK DEĞERLERİNE DEĞİŞİM)**



(MULTILAYER PERCEPTRON – 10 KAT ÇAPRAZ DOĞRULAMA BOYUNCA FARKLI KATLARIN EĞİTİM DOĞRULUKLARINI GÖSTEREN GRAFİK)

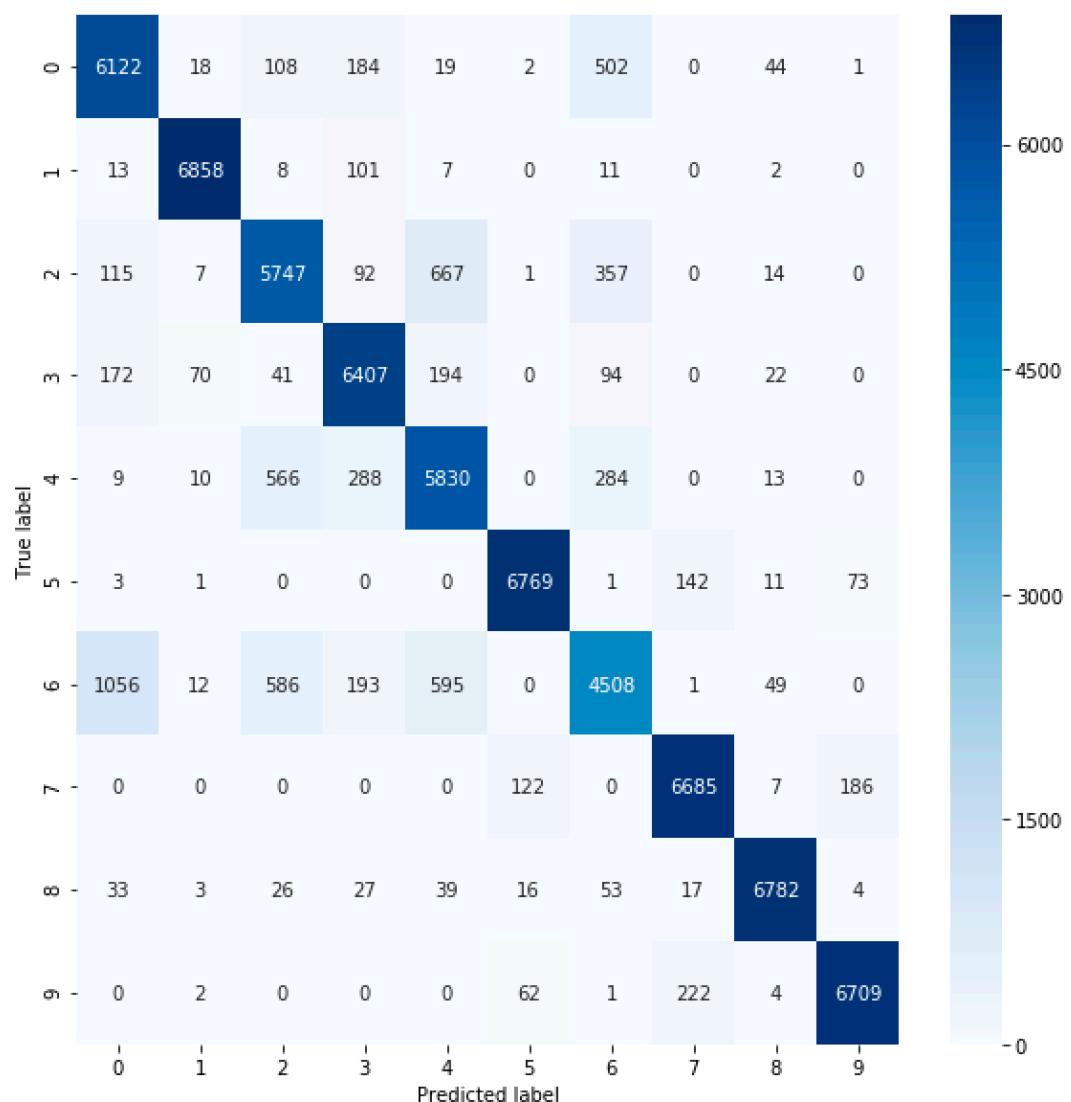


(MULTILAYER PERCEPTRON – 10 KAT ÇAPRAZ DOĞRULAMA BOYUNCA FARKLI KATLARIN TEST DOĞRULUKLARINI GÖSTEREN GRAFİK)

Accuracy Results										
FOLD NO: 1	Accuracy :	89.48571681976318								
FOLD NO: 2	Accuracy :	88.40000033378601								
FOLD NO: 3	Accuracy :	88.38571310043335								
FOLD NO: 4	Accuracy :	89.2285704612732								
FOLD NO: 5	Accuracy :	89.2714262008667								
FOLD NO: 6	Accuracy :	89.4428551197052								
FOLD NO: 7	Accuracy :	89.60000276565552								
FOLD NO: 8	Accuracy :	89.52857255935669								
FOLD NO: 9	Accuracy :	89.01428580284119								
FOLD NO: 10	Accuracy :	89.31428790092468								

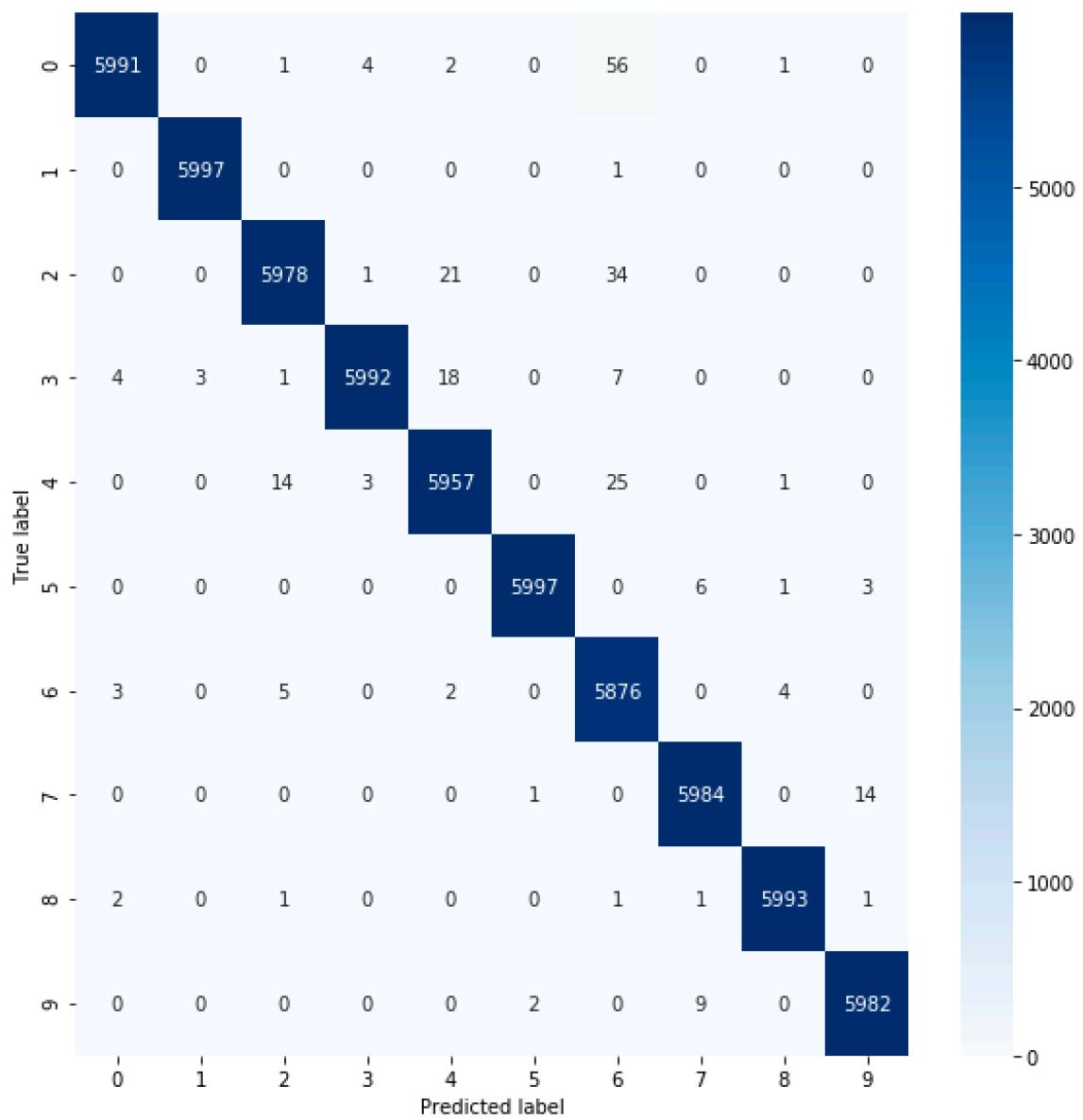
(MULTILAYER PERCEPTRON – 10 KATIN HER BİRİ İÇİN TEST SETİNE BAĞLI DOĞRULUK ORANLARI)

HATA MATRİSİ :



(MULTILAYER PERCEPTRON – TÜM KATLAR İÇİN SENTEZLENMİŞ HATA MATRİSİ)

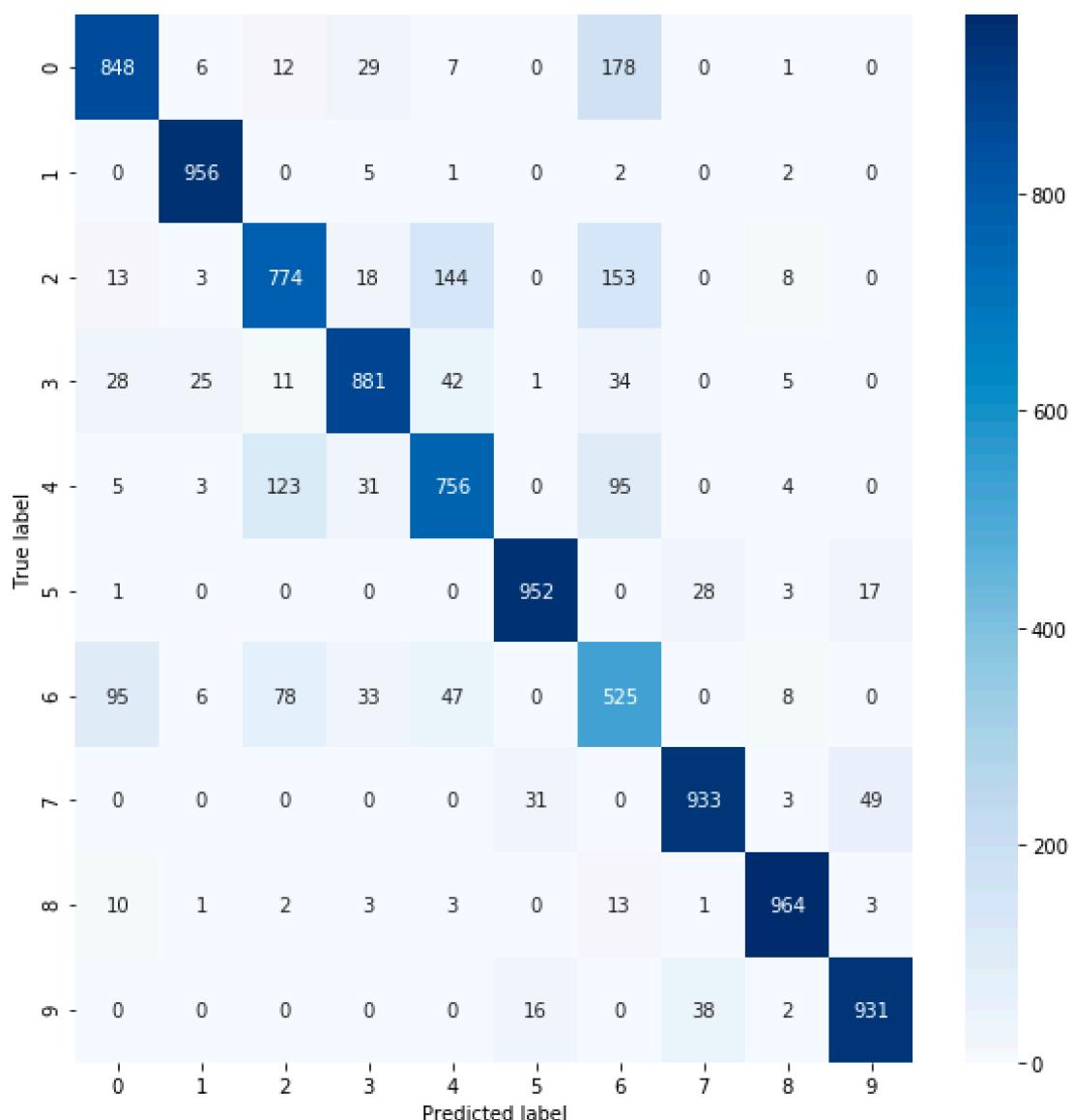
EĞİTİM VERİLERİ HATA MATRİSİ :



TRANINING ACCURACY : 0.995783333333334

(RANDOM FOREST – ÇAPRAZ DOĞRULAMA OLmadan EĞİTİM VERİLERİ İÇİN HATA MATRİSİ VE DOĞRULUK ORANI)

TEST VERİLERİ HATA MATRİSİ :

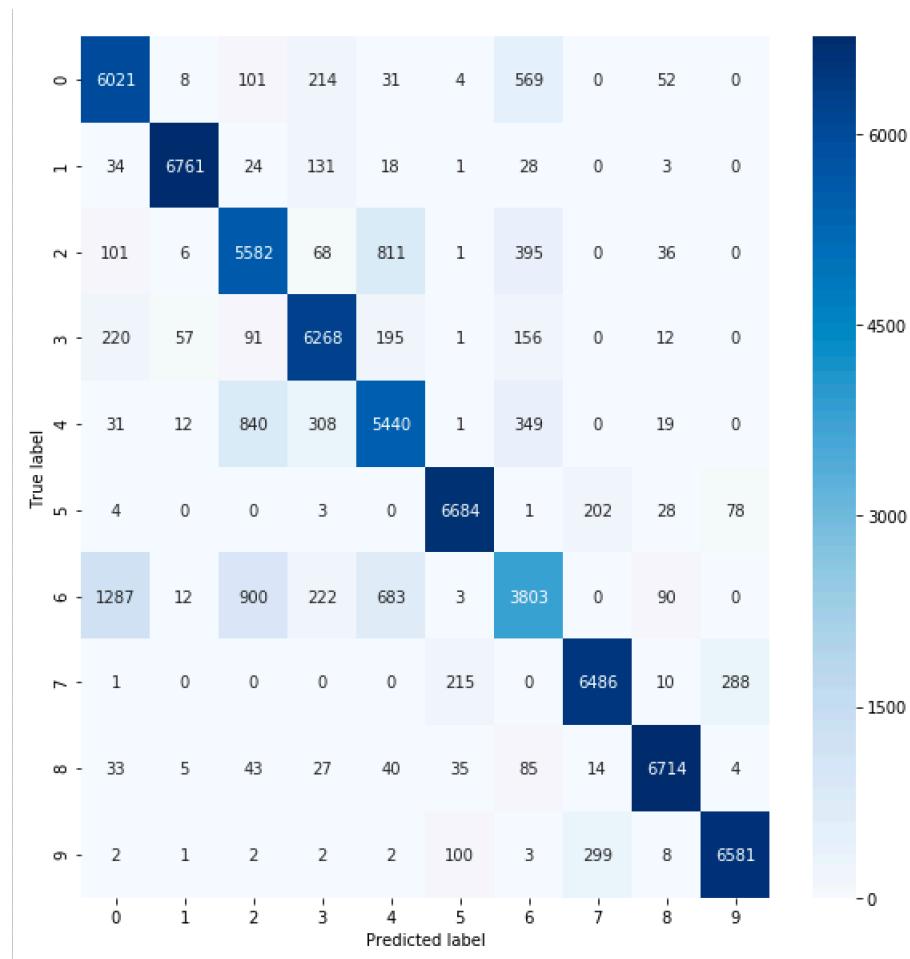


TEST ACCURACY : 0.852

(RANDOM FOREST- ÇAPRAZ DOĞRULAMA OLmadan TEST VERİLERİ İÇİN HATA MATRİSİ
VE DOĞRULUK ORANI)

AVERAGE ACCURACY : 86.19999999999999
STANDARD DEVIATION (+/-) : 0.2781865473060127
FOLD NO: 1 Accuracy : 86.55714285714285
FOLD NO: 2 Accuracy : 86.34285714285714
FOLD NO: 3 Accuracy : 86.32857142857144
FOLD NO: 4 Accuracy : 85.91428571428571
FOLD NO: 5 Accuracy : 86.45714285714286
FOLD NO: 6 Accuracy : 86.02857142857144
FOLD NO: 7 Accuracy : 86.24285714285715
FOLD NO: 8 Accuracy : 86.17142857142858
FOLD NO: 9 Accuracy : 85.57142857142857
FOLD NO: 10 Accuracy : 86.38571428571429

(RANDOM FOREST – 10 KAT ÇAPRAZ DOĞRULAMA UYGULANDIKTAN SONRA HER KAT İÇİN TEST SETİNE BAĞLI DOĞRULUK ORANLARI, ORTALAMA VE STANDART SAPMA DEĞERLERİ)



(RANDOM FOREST – 10 KAT ÇAPRAZ DOĞRULAMA UYGULANDIKTAN SONRA HER KATA AİT DOĞRULUK DEĞERLERİNİN SENTEZLENMESİ SONUCU OLUŞTURULMUŞ HATA MATRİSİ)

Doğruluk Değerleri (Accuracy) Tabloları

Multilayer Perceptron / MLP

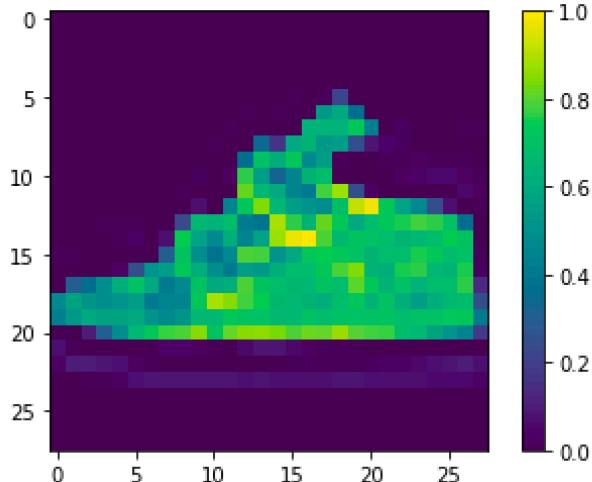
Fold	Accuracy
1	%89.48
2	%88.40
3	%88.39
4	%89.23
5	%89.27
6	%89.44
7	%89.60
8	%89.53
9	%89.01
10	%89.31

Random Forest

Fold	Accuracy
1	%86.56
2	%86.34
3	%86.32
4	%85.91
5	%86.46
6	%86.03
7	%86.24
8	%86.17
9	%85.57
10	%86.39

4.3 Kullanıcı tarafından verilen örneğin sınıflandırma ekran görüntüsü (konsol çıktıları)

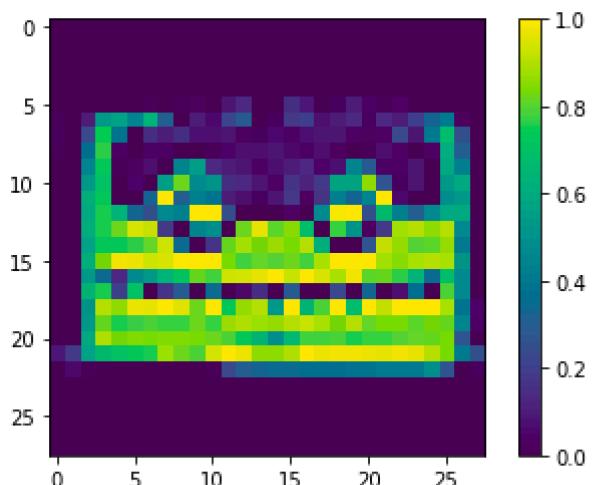
THE IMAGE THAT WILL GET IT'S LABEL PREDICTED:



THE PREDICTION IS: Sneaker
THE REAL CLASS IS: Sneaker

(RANDOM FOREST - RASGELE SEÇİLEREK VERİLEN ÖRNEĞİN SINIFLANDIRILMASI, 1. DENEME)

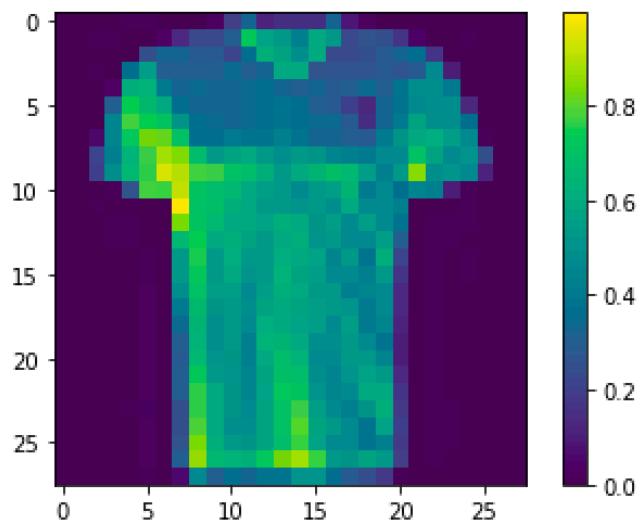
THE IMAGE THAT WILL GET IT'S LABEL PREDICTED:



THE PREDICTION IS: Bag
THE REAL CLASS IS: Bag

(RANDOM FOREST – RASGELE SEÇİLEREK VERİLEN ÖRNEĞİN SINIFLANDIRILMASI, 2. DENEME)

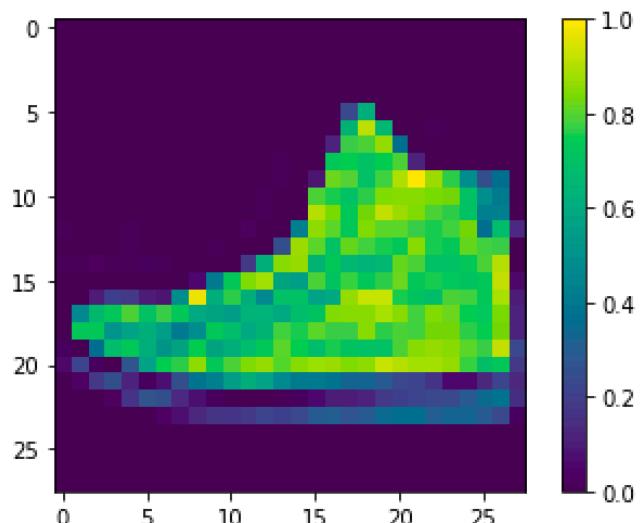
THE IMAGE THAT WILL GET IT'S LABEL PREDICTED:



THE PREDICTION IS: T-shirt
THE REAL CLASS IS: T-shirt

(RANDOM FOREST – RASGELE SEÇİLEREK VERİLEN ÖRNEĞİN SINIFLANDIRILMASI, 3. DENEME)

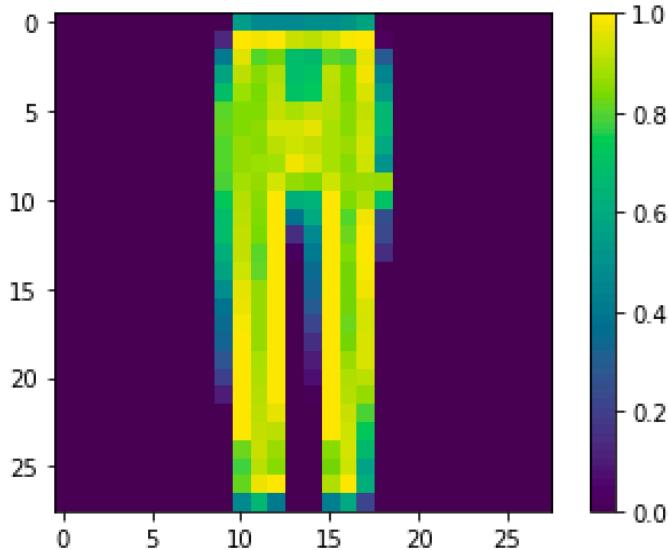
THE IMAGE THAT WILL GET IT'S LABEL PREDICTED:



THE PREDICTION IS: Ankle Boot
THE REAL CLASS IS: Sneaker

(RANDOM FOREST – RASGELE SEÇİLEREK VERİLEN ÖRNEĞİN SINIFLANDIRILMASI, HATALI SINIFLANDIRMA ÖRNEĞİ)

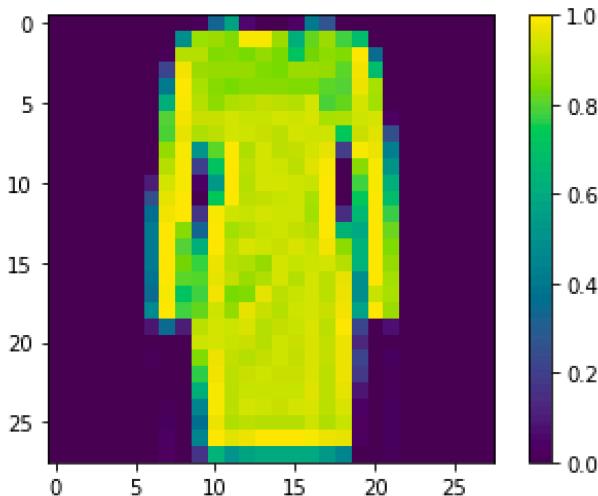
.... print(THE REAL CLASS IS: , class_name)
THE IMAGE THAT WILL GET IT'S LABEL PREDICTED:



THE PREDICTION IS: Trouser
THE REAL CLASS IS: Trouser

(MULTILAYER PERCEPTRON – RASGELE SEÇİLEREK VERİLEN ÖRNEĞİN SINIFLANDIRILMASI,
1. DENEME)

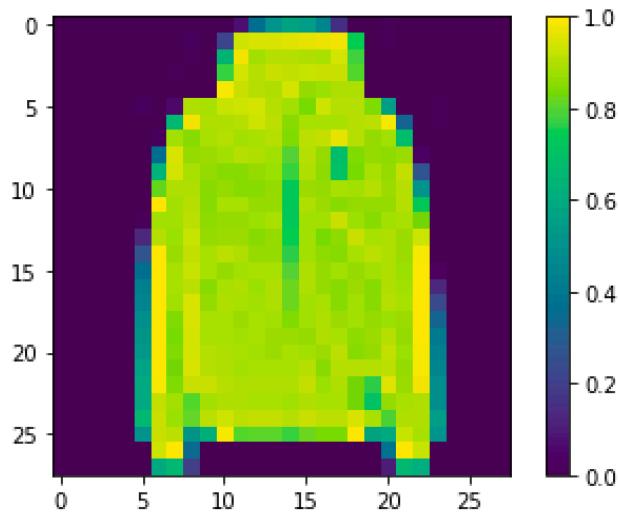
THE IMAGE THAT WILL GET IT'S LABEL PREDICTED:



THE PREDICTION IS: Dress
THE REAL CLASS IS: Dress

(MULTILAYER PERCEPTRON – RASGELE SEÇİLEREK VERİLEN BİR ÖRNEĞİN
SINIFLANDIRILMASI, 2.DENEME)

THE IMAGE THAT WILL GET IT'S LABEL PREDICTED:

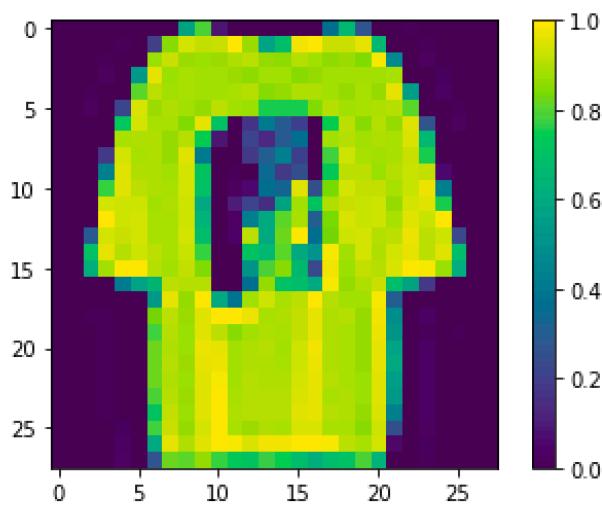


THE PREDICTION IS: Coat

THE REAL CLASS IS: Coat

(MULTILAYER PERCEPTRON – RASGELE SEÇİLEREK VERİLEN BİR ÖRNEĞİN SINIFLANDIRILMASI, 3. DENEME)

THE IMAGE THAT WILL GET IT'S LABEL PREDICTED:



THE PREDICTION IS: Shirt

THE REAL CLASS IS: T-shirt

(MULTILAYER PERCEPTRON – RASGELE SEÇİLEREK VERİLEN BİR ÖRNEĞİN SINIFLANDIRILMASI, HATALI SINIFLANDIRMA ÖRNEĞİ)

4.4 Kod ve Çeşitli Çıktı Görüntüleri

Kod Görüntüleri:

Random Forest :

```
#Gerekli kütüphaneler import edilir
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import StratifiedKFold
from tensorflow import keras
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
import random as r
import tensorflow as tf
```

```
| #Çapraz doğrulama fold sayısı belirlenir
n_splits = 10

#Veri seti yüklenir, gerekli eğitim ve test verilerine ayırtırılır.
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

#Veride mevcut olan sınıf isimlerini ifade eden bir liste tanımlanır.
class_names = ["T-shirt",
                "Trouser",
                "Pullover",
                "Dress",
                "Coat",
                "Sandal",
                "Shirt",
                "Sneaker",
                "Bag",
                "Ankle Boot"]
```

```
#Eğitim ve test verilerinin ve labellarının shape'leri kontrol edilir.
print("TRAIN DATA SHAPE : ", train_images.shape)
print("TRAIN LABELS LENGTH : ", len(train_labels))

print("TEST DATA SHAPE : ", test_images.shape)
print("TEST LABELS LENGTH : ", len(test_labels))

#Örnek birkaç veri görüntülenir
print("\n\n EXAMPLE DATA SAMPLES: \n")
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()

plt.figure()
plt.imshow(train_images[10])
plt.colorbar()
plt.grid(False)
plt.show()

plt.figure()
plt.imshow(train_images[105])
plt.colorbar()
plt.grid(False)
plt.show()
```

```
#Daha verimli bir eğitim için eğitim ve test verilerine normalizasyon uygulanır.
#Resimlerimiz siyah beyaz olduğu için ve siyah beyaz değerler 0 ve 255 arasında tanımlandığı için 255.0'e böldük.
train_images = train_images / 255.0
test_images = test_images / 255.0

#Çeşitli sınıflara ait verilere örnekler
print("\n EXAMPLES FROM VARIOUS CLASSES : \n")
plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()

#Eğitim ve test verisini düzleştir.
train_pixels = []
test_pixels = []
for i in range(0, len(train_images)):
    train_pixels.append(train_images[i].flatten().reshape(28*28))
for i in range(0, len(test_images)):
    test_pixels.append(test_images[i].flatten().reshape(28*28))
```

```
#Random Forest modelini eğit.  
rf_model = RandomForestClassifier().fit(train_pixels, train_labels)  
  
#Eğitim verileri için tahminlemeler üret  
predictions = rf_model.predict(train_pixels)  
  
#Eğitim verileri için confusion matrix oluştur  
con_matrix = confusion_matrix(predictions, train_labels)  
  
#Eğitim verileri için oluşturulan confusion matrixi görüntüle  
print("\n EĞİTİM VERİLERİ HATA MATRİSİ : \n")  
figure = plt.figure(figsize=(8, 8))  
sns.heatmap(con_matrix, annot=True, cmap=plt.cm.Blues, fmt='g')  
plt.tight_layout()  
plt.ylabel('True label')  
plt.xlabel('Predicted label')  
plt.show()  
  
#Eğitim verileri için accuracy oranını görüntüle  
rf_train_accuracy = (predictions == train_labels).mean()  
print("\n\n TRAINING ACCURACY : ", rf_train_accuracy, "\n\n")  
  
#Test verileri için tahminlemeler üret  
predictions = rf_model.predict(test_pixels)  
  
#Test verileri için confusion matrix oluştur  
con_matrix = confusion_matrix(predictions, test_labels)
```

```
#Test verileri için oluşturulan confusion matrixi görüntüle  
print("\n TEST VERİLERİ HATA MATRİSİ : \n")  
figure = plt.figure(figsize=(8, 8))  
sns.heatmap(con_matrix, annot=True, cmap=plt.cm.Blues, fmt='g')  
plt.tight_layout()  
plt.ylabel('True label')  
plt.xlabel('Predicted label')  
plt.show()  
  
#Test verileri için accuracy oranını görüntüle  
rf_test_accuracy = (predictions == test_labels).mean()  
print("\n\n TEST ACCURACY : ", rf_test_accuracy , "\n\n")
```

```

#Verilen bir veriye göre tahminlemede bulun
idx = r.randint(0, len(test_images)-1)
data_image = test_images[idx]
data_pixels = data_image.flatten().reshape(28*28)
data_answer = test_labels[idx]
data_pixels = np.array([data_pixels])
result = rf_model.predict(data_pixels)

print("THE IMAGE THAT WILL GET IT'S LABEL PREDICTED: ")
plt.figure()
plt.imshow(data_image)
plt.colorbar()
plt.grid(False)
plt.show()

print("THE PREDICTION IS: ", class_names[result[0]])
print("THE REAL CLASS IS: ", class_names[data_answer])

```

```

#k-Fold Çapraz doğrulama uygulamadan önce veri setlerini birleştir
inputs = np.concatenate((train_images, test_images))
input_pixels = np.concatenate((train_pixels, test_pixels))
targets = np.concatenate((train_labels, test_labels))

#10 katlı çapraz doğrulama uygula
cvscores = []
y_preds = []
y_trues = []
kfolds = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=0)
for train, test in kfolds.split(inputs, targets):

    print("\n\n TRAINING NEW FOLD... \n\n")

    rf_model = RandomForestClassifier().fit(input_pixels[train], targets[train])

    train_predictions = rf_model.predict(input_pixels[test])
    test_predictions = rf_model.predict(input_pixels[test])

    #her katta model performansını ölçümle
    rf_train_accuracy = (train_predictions == targets[test]).mean()
    print("\n TRAINING ACCURACY : ", rf_train_accuracy, "\n")
    rf_test_accuracy = (test_predictions == targets[test]).mean()
    print("\n TEST ACCURACY : ", rf_test_accuracy , "\n")

    cvscores.append(rf_test_accuracy*100)

    y_pred = []
    for i in range(0, len(test_predictions)):
        y_pred.append(test_predictions[i])

    y_preds = np.concatenate((y_preds, y_pred), axis=None)
    y_trues = np.concatenate((y_trues, targets[test]), axis=None)

print("AVERAGE ACCURACY : ", np.mean(cvscores))
print("STANDARD DEVIATION (+/-) : ", np.std(cvscores))

#Her tur için ulaşılan final accuracy skorlarını göster (test verileri için)
for i in range(0, n_splits):
    print("FOLD NO:",(i+1), " Accuracy : ", cvscores[i])

```

```
len(y_trues)
len(y_preds)
#Tüm toplam tahminlemeler için hata matrisini görüntüle
con_mat = tf.math.confusion_matrix(
    labels = y_trues,
    predictions = y_preds,
).numpy()

#Hata matrisini ekrana bastır
print("\n HATA MATRİSİ : \n")
figure = plt.figure(figsize=(8, 8))
sns.heatmap(con_mat, annot=True, cmap=plt.cm.Blues, fmt='g')
plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

Multilayer Perceptron / MLP:

```
0
1
2 #Gerekli kütüphaneler import edilir
3 import tensorflow as tf
4 from tensorflow import keras
5 import numpy as np
6 import matplotlib.pyplot as plt
7 import seaborn as sns
8 from sklearn.model_selection import StratifiedKFold
9 import mlp_model
10 import random as r
```

```
1
2
3 #batch boyutu, çapraz doğrulama ayrımlı sayısı ve eğitim nesli sayısı tanımlanır
4 batch_size = 64
5 epochs = 10
6 n_splits = 10
7
8
9 #Veri seti yüklenir, gerekli eğitim ve test verilerine ayırtırılır.
10 fashion_mnist = keras.datasets.fashion_mnist
11 (train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
12
13
```

```
14
15 #Veride mevcut olan sınıf isimlerini ifade eden bir liste tanımlanır.
16 class_names = ["T-shirt",
17                 "Trouser",
18                 "Pullover",
19                 "Dress",
20                 "Coat",
21                 "Sandal",
22                 "Shirt",
23                 "Sneaker",
24                 "Bag",
25                 "Ankle Boot"]
26
27
28 #Eğitim ve test verilerinin ve labellarının shape'leri kontrol edilir.
29 print("TRAIN DATA SHAPE : ", train_images.shape)
30 print("TRAIN LABELS LENGTH : ", len(train_labels))
31
32 print("TEST DATA SHAPE : ", test_images.shape)
33 print("TEST LABELS LENGTH : ", len(test_labels))
```

```
#Örnek birkaç veri görüntülenir
print("\n\n EXAMPLE DATA SAMPLES: \n")
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()

plt.figure()
plt.imshow(train_images[10])
plt.colorbar()
plt.grid(False)
plt.show()

plt.figure()
plt.imshow(train_images[105])
plt.colorbar()
plt.grid(False)
plt.show()

#Daha verimli bir eğitim için eğitim ve test verilerine normalizasyon uygulanır.
#Resimlerimiz siyah beyaz olduğu için ve siyah beyaz değerler 0 ve 255 arasında tanımlandığı için 255.0'e böldük.
train_images = train_images / 255.0
test_images = test_images / 255.0
```

```
#Çeşitli sınıflara ait verilere örnekler
print("\n EXAMPLES FROM VARIOUS CLASSES : \n")
plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()

##Modeli tanımladığı sınıfından edin
model = mlp_model.get_model()

##Model özetini görüntüle
print(model.summary())

##Model eğitimi başlat. (10-Fold Cross Validation aşağıda yapılacak.)
history = model.fit(
    train_images,
    train_labels,
    batch_size=batch_size,
    epochs=epochs,
    validation_data=(test_images, test_labels))

##Modeli değerlendir.
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
```

```

}
#Modelin test seti üzerindeki başarısını görüntüle
print("\n\n TEST ACCURACY: %", test_acc*100)

#
#Modele dayalı tahminlemelerde bulun
probability_model = tf.keras.Sequential([model,
                                         tf.keras.layers.Softmax()])

predictions = probability_model.predict(test_images)

y_pred = []
for i in range(0, len(predictions)):
    y_pred.append(np.argmax(predictions[i]))

##Modele yönelik gerçekleştirilen tahminlerden çıkarılan hata matrisini görüntüle
con_mat = tf.math.confusion_matrix(
    labels = test_labels,
    predictions = y_pred,
).numpy()

#
#Hata matrisini ekrana bastır
print("\n HATA MATRİSİ : \n")
figure = plt.figure(figsize=(8, 8))
sns.heatmap(con_mat, annot=True,cmap=plt.cm.Blues, fmt='g')
plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

```

```

#Eğitim ve test setine bağlı nesile göre değişen accuracy değerleri tarihini ekrana bastır.
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Accuracy History (Training and Validation)')
plt.ylabel('Accuracy Value (%)')
plt.xlabel('Number of Epoch')
plt.show()

#Verilen bir veriye göre tahminlemede bulun
idx = r.randint(0, len(test_images)-1)
data_image = test_images[idx]
data_answer = test_labels[idx]
data_image = np.array([data_image])
result = model.predict(data_image)
result = np.argmax(result)

print("THE IMAGE THAT WILL GET IT'S LABEL PREDICTED: ")
plt.figure()
plt.imshow(data_image[0])
plt.colorbar()
plt.grid(False)
plt.show()

print("THE PREDICTION IS: ", class_names[result])
print("THE REAL CLASS IS: ", class_names[data_answer])

```

```
#k-Fold Çapraz doğrulama uygulamadan önce veri setlerini birleştir
inputs = np.concatenate((train_images, test_images))
targets = np.concatenate((train_labels, test_labels))

#10 katlı çapraz doğrulama uygula
cvscores = []
histories = []
y_preds = []
y_trues = []
kfolds = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=0)
for train, test in kfolds.split(inputs, targets):

    print("\n\n TRAINING NEW FOLD... \n\n")

    model = mlp_model.get_model()

    history = model.fit(inputs[train],
                         targets[train],
                         batch_size=batch_size,
                         epochs=epochs,
                         validation_data=(inputs[test], targets[test]))

    #her katta model performansını ölçümle
    scores = model.evaluate(inputs[test], targets[test], verbose=0)
    print("\n\n TEST ACCURACY : %", scores[1]*100)

    cvscores.append(scores[1]*100)
    histories.append(history)

    #modele bağlı tahminlemelerde bulun
    probability_model = tf.keras.Sequential([model,
                                              tf.keras.layers.Softmax()])

    predictions = probability_model.predict(inputs[test])

    y_pred = []
    for i in range(0, len(predictions)):
        y_pred.append(np.argmax(predictions[i]))

    y_preds = np.concatenate((y_preds, y_pred), axis=None)
    y_trues = np.concatenate((y_trues, targets[test]), axis=None)

print("AVERAGE ACCURACY : ", np.mean(cvscores))
print("STANDARD DEVIATION (+/-) : ", np.std(cvscores))
```

```
3 #Her kat için accuracy değerinin zamana bağlı değişimini göster (eğitim verisi için)
4 for i in range(0, len(histories)):
5     plt.plot(histories[i].history['accuracy'])
6 plt.title('Training Accuracy History')
7 plt.ylabel('Accuracy Value (%)')
8 plt.xlabel('Number of Epoch')
9 plt.show()
10
11 #Her kat için accuracy değerinin zamana bağlı değişimini göster (test verisi için)
12 for i in range(0, len(histories)):
13     plt.plot(histories[i].history['val_accuracy'])
14 plt.title('Validation Accuracy History')
15 plt.ylabel('Accuracy Value (%)')
16 plt.xlabel('Number of Epoch')
17 plt.show()
18
19 #Her tur için ulaşılan final accuracy skorlarını göster (test verileri için)
20 for i in range(0, n_splits):
21     print("FOLD NO:",(i+1), " Accuracy : ", cvscores[i])
22
23 #Tüm toplam tahminlemeler için hata matrisini görüntüle
24 con_mat = tf.math.confusion_matrix(
25     labels = y_trues,
26     predictions = y_preds,
27     ).numpy()
28
29
30 #Hata matrisini ekrana bastır
31 print("\n HATA MATRİSİ : \n")
32 figure = plt.figure(figsize=(8, 8))
33 sns.heatmap(con_mat, annot=True,cmap=plt.cm.Blues, fmt='g')
34 plt.tight_layout()
35 plt.ylabel('True label')
36 plt.xlabel('Predicted label')
37 plt.show()
```

Kullanılan Perceptron Modelinin Mimarısını içeren Kod:

```
import tensorflow as tf
from tensorflow import keras

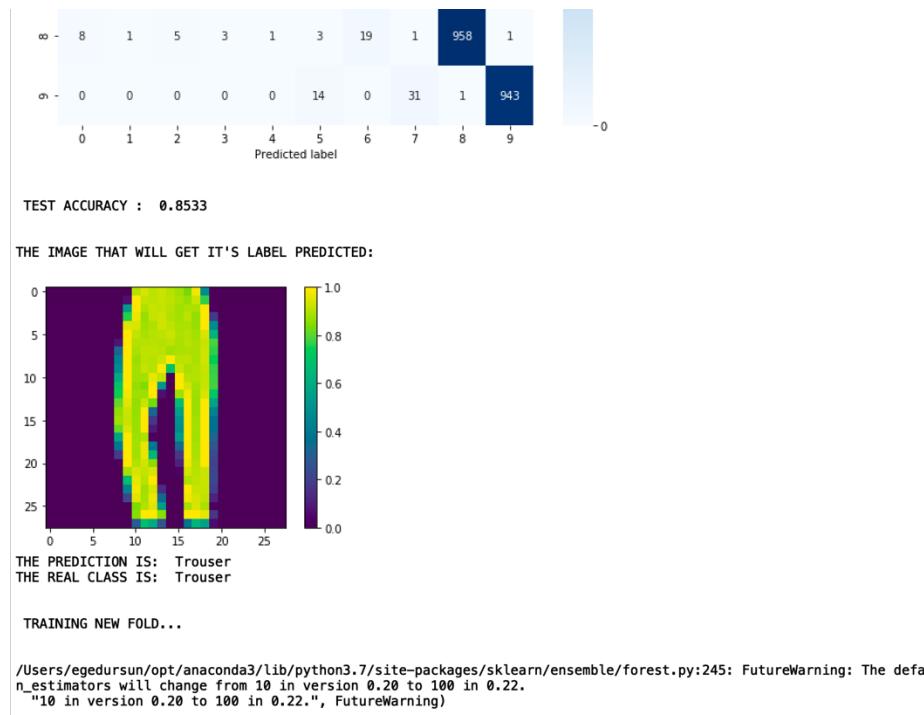
#Multilayer perceptron modelinin tanımı.
# FLATTEN -> DENSE -> DROPOUT -> DENSE -> DROPOUT -> DENSE -> DENSE
def get_model():
    model = keras.Sequential([
        keras.layers.Flatten(input_shape=(28, 28)),
        keras.layers.Dense(512, activation='relu'),
        keras.layers.Dropout(0.1),
        keras.layers.Dense(256, activation="relu"),
        keras.layers.Dropout(0.1),
        keras.layers.Dense(128, activation="relu"),
        keras.layers.Dense(10),
    ])

    model.compile(optimizer='adam',
                  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  metrics=["accuracy"])

    return model
```

Kod Çıktıları:

Random Forest:



```
TRAINING NEW FOLD...

/Users/egedursun/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/forest.py:245: FutureWarning: The def
n_estimators will change from 10 in version 0.20 to 100 in 0.22.
"10 in version 0.20 to 100 in 0.22.", FutureWarning)

TRANINING ACCURACY : 0.8602857142857143

TEST ACCURACY : 0.8602857142857143

TRAINING NEW FOLD...

/Users/egedursun/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/forest.py:245: FutureWarning: The def
n_estimators will change from 10 in version 0.20 to 100 in 0.22.
"10 in version 0.20 to 100 in 0.22.", FutureWarning)

TRANINING ACCURACY : 0.8574285714285714

TEST ACCURACY : 0.8574285714285714

TRAINING NEW FOLD...

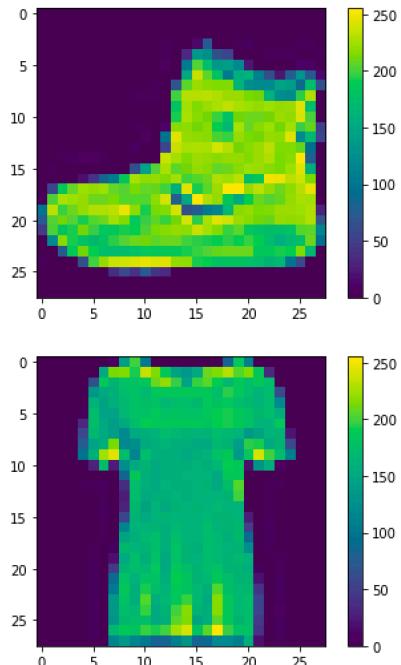
/Users/egedursun/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/forest.py:245: FutureWarning: The def
n_estimators will change from 10 in version 0.20 to 100 in 0.22.
"10 in version 0.20 to 100 in 0.22.", FutureWarning)

TRANINING ACCURACY : 0.8588571428571429

TEST ACCURACY : 0.8588571428571429
```

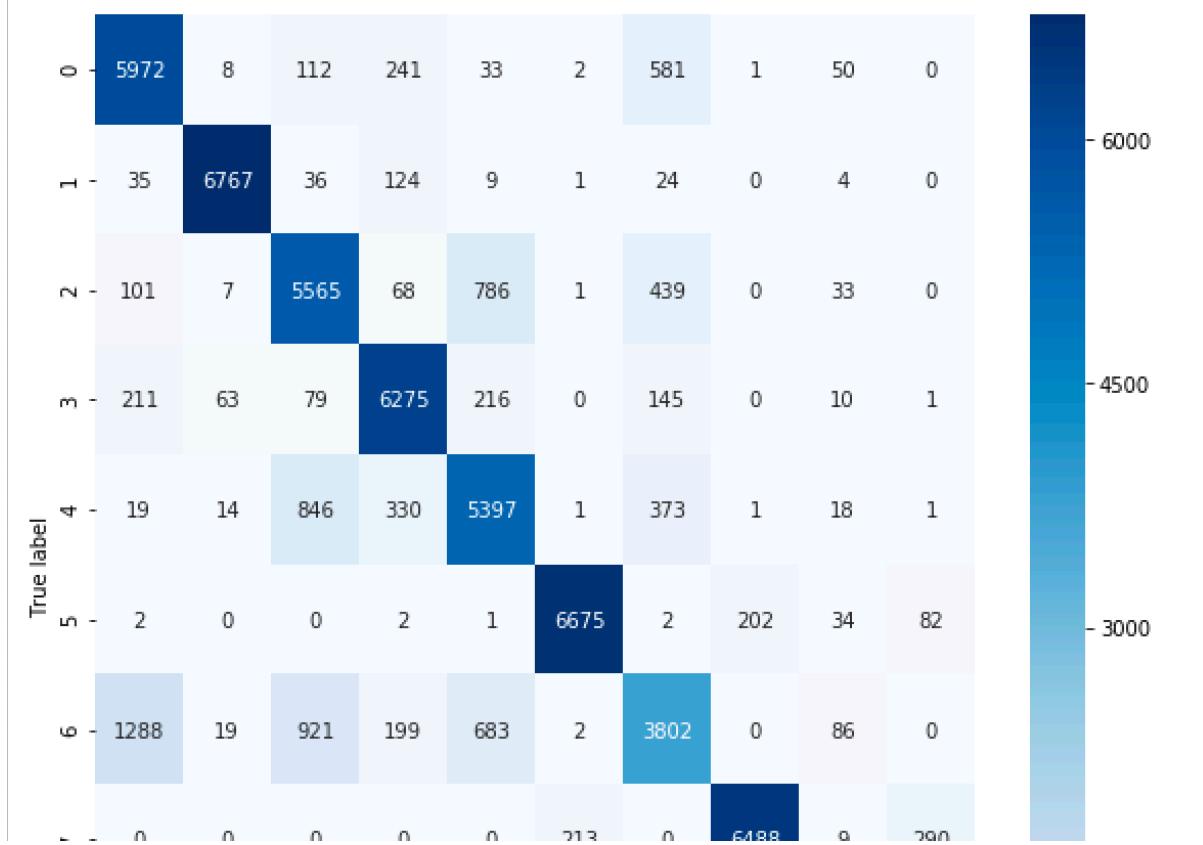
```
RELOADED MODULES: __mp_context, __multiprocessing
TRAIN DATA SHAPE : (60000, 28, 28)
TRAIN LABELS LENGTH : 60000
TEST DATA SHAPE : (10000, 28, 28)
TEST LABELS LENGTH : 10000
```

EXAMPLE DATA SAMPLES:



AVERAGE ACCURACY : 86.02428571428571
STANDARD DEVIATION (+/-) : 0.3010898570872435
FOLD NO: 1 Accuracy : 86.02857142857144
FOLD NO: 2 Accuracy : 85.74285714285715
FOLD NO: 3 Accuracy : 85.88571428571429
FOLD NO: 4 Accuracy : 85.88571428571429
FOLD NO: 5 Accuracy : 86.11428571428571
FOLD NO: 6 Accuracy : 85.5142857142857
FOLD NO: 7 Accuracy : 86.68571428571428
FOLD NO: 8 Accuracy : 86.15714285714286
FOLD NO: 9 Accuracy : 85.95714285714286
FOLD NO: 10 Accuracy : 86.27142857142857

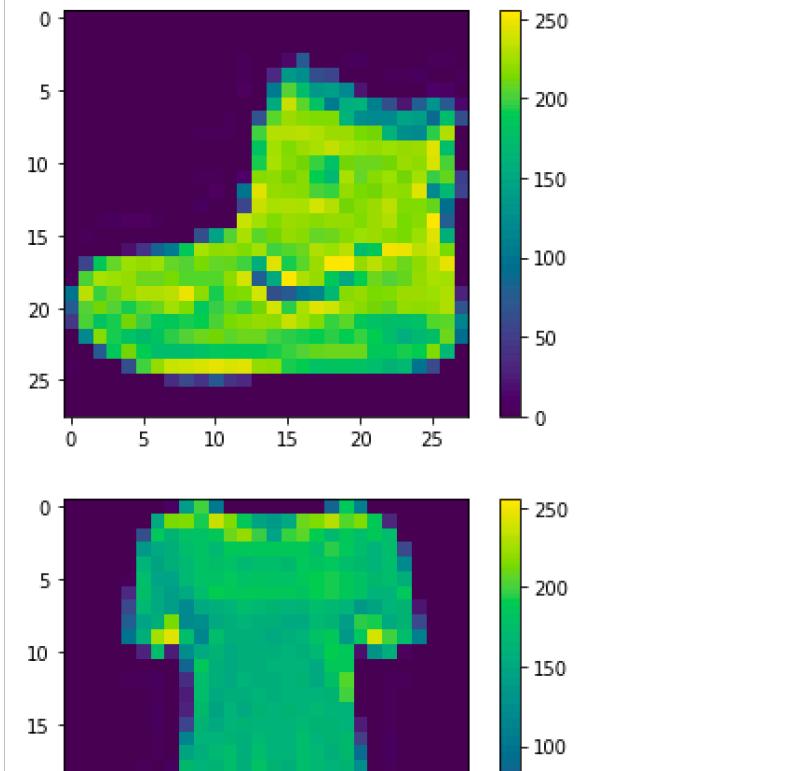
HATA MATRİSİ :



Multilayer Perceptron (MLP):

```
Proje 1/Soru 4)
TRAIN DATA SHAPE : (60000, 28, 28)
TRAIN LABELS LENGTH : 60000
TEST DATA SHAPE : (10000, 28, 28)
TEST LABELS LENGTH : 10000
```

EXAMPLE DATA SAMPLES:

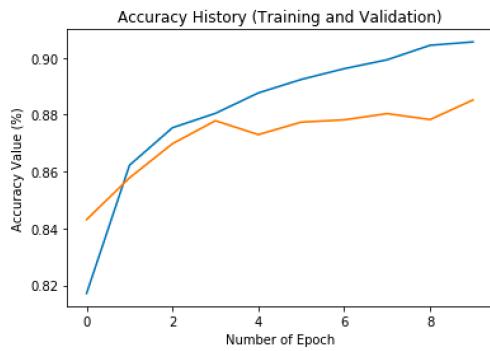
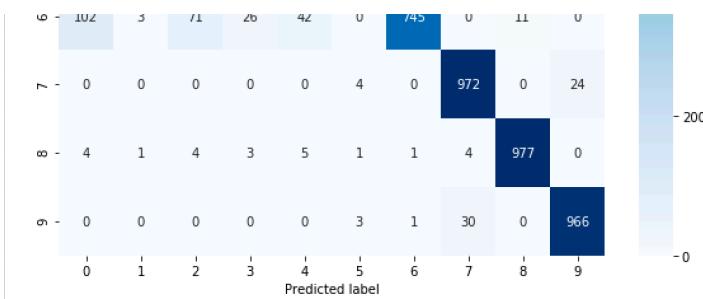


Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 784)	0
dense_4 (Dense)	(None, 512)	401920
dropout_2 (Dropout)	(None, 512)	0
dense_5 (Dense)	(None, 256)	131328
dropout_3 (Dropout)	(None, 256)	0
dense_6 (Dense)	(None, 128)	32896
dense_7 (Dense)	(None, 10)	1290
<hr/>		
Total params: 567,434		
Trainable params: 567,434		
Non-trainable params: 0		

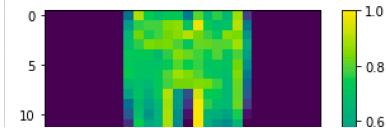
```

transformed and will be executed as-is. Please report this to the AutoGraph team. When filing the bug, set the verbosity to 10 (Linux, `export AUTOGRAPH_VERTOSITY=10`) and attach the full output. Cause:
60000/60000 [=====] - 6s 100us/sample - loss: 0.5051 - accuracy: 0.8171 - val_loss: 0.4317 - val_accuracy: 0.8431
Epoch 2/10
60000/60000 [=====] - 6s 92us/sample - loss: 0.3743 - accuracy: 0.8622 - val_loss: 0.3832 - val_accuracy: 0.8578
Epoch 3/10
60000/60000 [=====] - 5s 91us/sample - loss: 0.3404 - accuracy: 0.8754 - val_loss: 0.3638 - val_accuracy: 0.8698
Epoch 4/10
60000/60000 [=====] - 6s 92us/sample - loss: 0.3203 - accuracy: 0.8805 - val_loss: 0.3360 - val_accuracy: 0.8779
Epoch 5/10
60000/60000 [=====] - 6s 94us/sample - loss: 0.3021 - accuracy: 0.8877 - val_loss: 0.3516 - val_accuracy: 0.8730
Epoch 6/10
60000/60000 [=====] - 5s 91us/sample - loss: 0.2894 - accuracy: 0.8924 - val_loss: 0.3458 - val_accuracy: 0.8774
Epoch 7/10
60000/60000 [=====] - 5s 91us/sample - loss: 0.2767 - accuracy: 0.8962 - val_loss: 0.3319 - val_accuracy: 0.8782
Epoch 8/10
60000/60000 [=====] - 5s 91us/sample - loss: 0.2657 - accuracy: 0.8993 - val_loss: 0.3341 - val_accuracy: 0.8804
Epoch 9/10
60000/60000 [=====] - 5s 91us/sample - loss: 0.2574 - accuracy: 0.9044 - val_loss: 0.3321 - val_accuracy: 0.8783
Epoch 10/10
42560/60000 [=====>.....] - ETA: 1s - loss: 0.2491 - accuracy: 0.9061

```



THE IMAGE THAT WILL GET IT'S LABEL PREDICTED:



```
transformed and will be executed as-is. Please report this to the AutoGraph team. When filing the bug, set the verbosity to 10
Linux, `export AUTOGRAPH_VERTBOSITY=10`) and attach the full output. Cause:
63000/63000 [=====] - 6s 103us/sample - loss: 0.4946 - accuracy: 0.8199 - val_loss: 0.3925 - val_accuracy
0.8559
Epoch 2/10
63000/63000 [=====] - 6s 90us/sample - loss: 0.3792 - accuracy: 0.8609 - val_loss: 0.3542 - val_accuracy
0.8754
Epoch 3/10
63000/63000 [=====] - 6s 90us/sample - loss: 0.3426 - accuracy: 0.8734 - val_loss: 0.3455 - val_accuracy
0.8769
Epoch 4/10
63000/63000 [=====] - 6s 90us/sample - loss: 0.3202 - accuracy: 0.8812 - val_loss: 0.3156 - val_accuracy
0.8833
Epoch 5/10
28480/63000 [=====>.....] - ETA: 2s - loss: 0.3038 - accuracy: 0.8843
```

```
63000/63000 [=====] - 6s 91us/sample - loss: 0.3036 - accuracy: 0.8853 - val_loss: 0.3321 - val_accuracy
0.8773
Epoch 6/10
63000/63000 [=====] - 6s 88us/sample - loss: 0.2909 - accuracy: 0.8925 - val_loss: 0.3195 - val_accuracy
0.8804
Epoch 7/10
63000/63000 [=====] - 5s 87us/sample - loss: 0.2783 - accuracy: 0.8954 - val_loss: 0.2927 - val_accuracy
0.8969
Epoch 8/10
63000/63000 [=====] - 6s 87us/sample - loss: 0.2709 - accuracy: 0.8971 - val_loss: 0.3216 - val_accuracy
0.8856
Epoch 9/10
63000/63000 [=====] - 6s 89us/sample - loss: 0.2597 - accuracy: 0.9017 - val_loss: 0.2919 - val_accuracy
0.8959
Epoch 10/10
63000/63000 [=====] - 6s 87us/sample - loss: 0.2521 - accuracy: 0.9040 - val_loss: 0.3053 - val_accuracy
0.8934
```

```
TEST ACCURACY : % 89.34285640716553
```

```
TRAINING NEW FOLD...
```

```
Train on 63000 samples, validate on 7000 samples
Epoch 1/10
WARNING:tensorflow:Entity <function Function._initialize_uninitialized_variables.<locals>.initialize_variables at 0x1a4e84eef0> could not be transformed and will be executed as-is. Please report this to the AutoGraph team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERTBOSITY=10`) and attach the full output. Cause:
WARNING: Entity <function Function._initialize_uninitialized_variables.<locals>.initialize_variables at 0x1a4e84eef0> could not be transformed and will be executed as-is. Please report this to the AutoGraph team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERTBOSITY=10`) and attach the full output. Cause:
52288/63000 [=====>.....] - ETA: 0s - loss: 0.5163 - accuracy: 0.8140
```

5. Öz Değerlendirme

	İstenen Özellik	Var	Açıklama	Tahmini Not
1a	Algoritmalar + Karmaşıklıklar (10)	<input checked="" type="checkbox"/>	Sözü geçen algoritmalar ile ilgili detaylı araştırmada bulundum; sadece birinin karmaşıklık düzeyleriyle ilgili emin olamadım.	8
1b	Tanım ve Karşılaştırmalar (10)	<input checked="" type="checkbox"/>	Bahsedilen kavramlarla ilgili detaylı araştırma gerçekleştirdim, tanım ve karşılaştırmalarını çıkarımlarda bulunarak yaptım.	10
1c	Araştırma ve Yorum (10)	<input checked="" type="checkbox"/>	Etrafıca bir araştırmayla birden çok kaynaktan, sıkılıkla yapılan hataları öğrenip, derledim.	10
2	Problem Çözme ve Kodlama (10)	<input checked="" type="checkbox"/>	A* algoritmasına ek olarak Brute Force ile de istenen probleme çözüm sağlamak için bir kod geliştirdim. Kullanıcının ve kod okuyucusunun işini kolaylaştıracak görseller ve notlar ile kodu zenginleştirdim.	10
3	Minimax Algoritması (15)	<input checked="" type="checkbox"/>	Problemi zenginleştirerek günlük hayatı karşılaşabilecek bir forma getirdim ve kısıtlı zaman limiti durumunda kullanılabilecek bir özellik kazandırdım.	15
4	Makine Öğrenmesi (25)	<input checked="" type="checkbox"/>	İki farklı sınıflandırıcı kullanarak başarılı sayılabilen doğruluk oranları ile Fashion MNIST verisetini kullanarak tahminlemeler yapabildim.	25
	Rapor (20)	<input checked="" type="checkbox"/>	Okunabilir, açık ve ifade edici bir rapor hazırladığımı inanıyorum.	20
100 üzerinden Toplam Not:				98

Harcanan Süreler:

Tahmini 90-100 saat

Proje ekibi : Ege Doğan Dursun - 05170000006