T.C.
SAKARYA ÜNİVERSİTESİ
BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BSM 448
PRATİKTE BT VE BS UYGULAMALARI

ÖDEV 2

Hazırlayan

B141210306
Emre BODUR

Öğretim Üyesi

Prof. Dr. Nejat YUMUŞAK

DÜZCE MAYIS-2021

PRATİKTE BT VE BS UYGULAMALARI 2. ÖDEV CEVAPLARI

1. Python dili, "ScikitLearn" kütüphanesi ile Makine Öğrenmesi (Naïve Bayes ve İkili Arama Ağacı algoritmaları ile sınıflandırma) uygulaması gerçekleştiriniz. Mevcut kod üzerinde satır satır değişiklikler yaparak farklı algoritma ve parametrelerle tahmin yapma sistemi geliştirebilirsiniz.

Tanımlar

Bilgisayarların direkt programlanmadan, kendilerine insan gözlemlerinin bilgi ve veri formunda verilmesiyle, insanlar gibi davranıp öğrenmesi makine öğrenmesi olarak tanımlanmaktadır. Makine öğrenmesinde pek çok algoritma kullanılır. Bunlardan biri de Supervised Learning (Gözetimli öğrenme) başlığı altında incelediğimiz bir Classification (Sınıflandırma) olan Naive Bayes'tir.

Naive Bayes üretken(generative) bir modeldir. Sınıflandırma, veri setimizdeki ayırt etmemize yarayan belirli özelliklerine (features (X)) bakarak hedefimizi (target (y)) kategorilere ayırmamızı sağlar. Naïve Bayes sınıflandırıcı, örüntü tanıma problemine ilk bakışta oldukça kısıtlayıcı görülen bir önerme ile kullanılabilen olasılıksal bir yaklaşımdır.

Naive Bayes'in Gaussian Naive Bayes, Multinominal Naive Bayes ve Bernoulli Naive Bayes olmak üzere türleri vardır. Gaussian Naive Bayes Eğer özelliklerimiz sürekli değer (continuous value) ise bu değerlerin bir gauss dağılımı veya diğer bir değişle normal dağılımdan örneklendiğini varsaydığımız bir yaklaşımdır.

Scikit-learn, veri bilimi ve machine learning için en yaygın kullanılan Python paketlerinden biridir. Birçok işlemi gerçekleştirmenizi sağlar ve çeşitli algoritmalar sağlar. Scikit-learn ayrıca sınıfları, yöntemleri ve işlevleri ile kullanılan algoritmaların arka planıyla ilgili belgeler sunar.

Scikit-learn veri işleme, boyutsal küçülme, model seçimi, regresyon, sınıflandırılması, küme analizi özellikler bulunmaktadır. Ayrıca, modellerinizi test etmek için kullanabileceğiniz birkaç veri kümesi de sağlar (https://scikit-learn.org/stable/datasets.html adresinden datasetlere erişilebilir).

İkili arama ağacı, verileri organize etmek için kullanılan bir çeşit ikili ağaçtır. İkili ağaçtan temel farkı, verilerin sıralanmış bir şekilde tutulmasıdır, bu sayede ikili arama algoritmasının kullanılmasına imkân verir. Karar ağaçları ise, Sınıflandırma ve Regresyon problemlerinde kullanılan, ağaç tabanlı algoritmadan biridir. Karmaşık veri setlerinde kullanılabilir.

Uygulama:

Gaussian Naive Bayes Algoritması Örneği

Birinci soruyu gerçekleştirmek için öncelikle bilgisayarımızda Python kurulu olmalıdır. Editör olarak Visual Studio Code, PyCharm veya Jupyter Notebook kullanılabilir. Kullanım kolaylığı açısından ben Jupyter Notebook kullanmayı tercih ettim. Editörü açtıktan sonra ilk yapmamız gereken ilgili kütüphaneleri projemize dahil etmek olacaktır. Bunun için aşağıdaki kodları yazdım.

```
# Proje ile ilgili kütüphaneler dahil ediliyor.
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
from sklearn import tree
import pandas as pd
import joblib
```

Kütüphanelerimizi ekledikten sonra Scikit-learn kütüphanesinde bize sunulan veri kümelerinden birini seçerek projeme dahil ettim (https://scikit-learn.org/stable/datasets.html adresinden veri kümelerine erişilebilir). Bunun için aşağıdaki kodu yazdım. Eğer hazır veri setinin yerine kendi veri setimi kullanmak isteseydim. Verilerimi csv formatında kaydettikten sonra pandas kütüphanesi ile kolay bir şekilde okuyabilirdim.

```
# iris dataseti yükleniyor
from sklearn.datasets import load_iris
irisData = load_iris()
```

Yukarıdaki kod satırında Scikit-learn kütüphanesinde kullanıma sunulan Iris verisetini projeme dahil etmiş oldum. Veri setinde Iris çiçeğinin üç türüne (setosa, versicolor, virginica) ait 50'şer tane, toplamda 150 tane olmak üzere üst ve alt çiçek yapraklarını ölçümleri bulunmaktadır. Bu ölçümden dört nitelikli [sepal-length (alt yaprak uzunluğu cm), sepal-with (alt yaprak genişliği cm), pedal-length (üst yaprak genişliği cm), pedal-width (üst yaprak uzunluğu cm)] ve 150 elemanlı bir veri seti elde etmiş. Bu veri seti makine öğrenmesi (machine learning) alıştırmalarında çok sıklıkla kullanıla gelmiş bir veri setidir. Veri setinin ön işlemesi yapılmış ve yukarıdaki üç türü temsilen bir rakam atanmış. Setosa 0, versicolor 1 ve virginica 2.

Projeme veri setinin dahil edilmesinden sonra bağımlı ve bağımsız değişkenleri oluşturmak için aşağıdaki kod satırını yazdım. Böylece bağımsız nitelikleri X, bağımlı niteliği y değişkenine atadım.

```
X = irisData.data
y = irisData.target
```

İlk 5 kaydı görmek için pandas modülünü kullanarak dataframe oluşturuyorum. Daha sonra head fonksiyonu ile ilk 5 kaydı gösteriyorum.

```
# pandas modülü ile dataframe oluşturuluyor.
# iris dataseti içerisindeki ilk 5 kayıt gösteriliyor.
data_frame = pd.DataFrame(irisData.data, columns=irisData.feature_names)
data_frame.head()
```

Veri setimizin ilk 5 kaydı aşağıdaki gibidir.

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Veri seti içerisindeki verileri eğitim ve test veri setleri olarak ayırdım. Doğru sonuç almak için verinin %75'i eğitim için, % 25'i ise test için ayrıldı. Kod satırı şöyledir:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 0)
```

Veri seti üzerinde işlemlerimi bitirdim. Şimdi Gaussian Naive Bayes Modeli oluşturuyorum. (GaussianNB() ile ilgili detaylı bilgi https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB adresinden öğrenilebilir.)

```
# Gaussian Naive Bayes algroritması ile model nesnesi oluşturuyorum.
model = GaussianNB()
```

Oluşturduğumuz modeli verilerimiz ile eğitelim.

```
# Oluşturduğumuz modeli verilerimiz ile eğitelim.
model.fit(X_train, y_train)
```

Modelimizi eğittik. Uygulamamızda kullanacağımız veri miktarı büyüdüğünde modeli eğitmek zaman alabilir. Veya başarı oranı her zaman yakalanmayabilir. Bu nedenle modeli bir kez eğittikten sonra kaydederek kayıtlı modelden çalıştırmak faydalı olacaktır. Eğittiğimiz modeli dosyaya kaydedelim. Bunun için sklearn tarafından sunulan joblib modülünü kullanacağız.

```
joblib.dump(model, 'egitilmis_model_GaussianNB.joblib')
```

Eğittiğimiz modeli dosyadan yükleyelim. Eğitilen bir modeli kullanacağımız için yukarıdaki `model.fit(X_train, y_train)` satırını comment satırı yapmalıyız.

```
model = joblib.load('egitilmis_model_GaussianNB.joblib')
```

Test Seti ile tahmin yaptım ayırdığım test setimi (`X_test`) kullanarak oluşturduğum model ile tahmin yapalım ve elde ettiğimiz set (`y_pred`) ile hedef değişken (`y_test`) test setimizi karşılaştıralım.

```
y_pred = model.predict(X_test)
```

Eğer tek bir değer için tahmin yapmak istiyorsak şu kodu kullanabiliriz.

```
# Eğer tek bir değeri tahmin ettirmek istiyorsak
y_pred = model.predict([[5.8, 2.8, 5.1, 2.4]])
```

Tahmin Sonuçlarını Test Sonuçları ile Karşılaştırma

Hata matrisi (confusion matrix) kullanarak modelin başarısını ölçelim:

```
# Tahmin Sonuçlarını Test Sonuçları ile Karşılaştırma
# Hata matrisi (confusion matrix) kullanarak modelin başarısını ölçelim:
print('Karmaşıklık Matrisi:')
print(irisData.target_names)
print(confusion_matrix(y_test, y_pred))
```

Şekil 1. Karmaşıklık Matrisi Sonucu

```
#Karmaşıklık Matrisi:
['setosa' 'versicolor' 'virginica']
[[13  0  0]
 [ 0 16  0]
 [ 0  0  9]]
```

Başarı Oranının Hesaplanması

Başarı oranını (accuracy_score) hesaplayarak ekrana yazdıralım.

```
# Başarı Oranı
# test_size = 0.25 ile GaussianNB algoritmasını kullanarak %100 oranında başarı ile çalıştı.
print ("Doğruluk (%) : ", 100*accuracy_score(y_test, y_pred))
Doğruluk (%) :  100.0
```

Yorum:

Matrisin köşegenlerin toplamı bize test edilen kayıt sayısını vermektedir. Karmaşıklık matrisinin köşegenleri toplamı 38'dir. Bu noktadan hareketle karmaşık matrisini yorumlayalım: 38 tane test kaydında Setosa sınıfına ait 13 tane kayıt varmış ve hepsi doğru tahmin edilmiş. 16 tane versicolor varmış bunların tamamı doğru tahmin edilmiş. 9 tane virginica varmış ve hepsi de başarıyla virginica olarak tahmin edilmiş.

Şekil 2. Karmaşıklık Matrisi

```
[[13  0  0]
 [ 0 16  0]
 [ 0  0  9]]
```

Başarı oranını yorumlayacak olursak, projemizde Iris veri setinde `test_size = 0.25` ile `GaussianNB` sınıfını kullanarak modelimizi eğittik sonrasında ise tahminlerde bulunmasını istedik. Bu tahminler sonucunda başarı oranını 1 olarak buldu. Bu skor tam ve doğru bir başarı yakaladığımızı (%100 başarı) göstermektedir. Eğer `test_size` değişkeninin değeri değiştirilerek program yeniden çalıştırılır ise farklı sonuçlar elde edilebilir.

Decision Tree Classifier Algoritması Örneği

Yukarıda *Gaussian Naive Bayes* algoritması ile bir örnek yaptık. Şimdi *Decision Tree Classifier* algoritması ile bir örnek uygulama yapalım. Bu örnekte de kütüphanelerin dahil edilmesi, Iris veri setinin yüklenmesi ve veri seti içerisindeki verilerin eğitim ve test verisi olarak ayrılması işlemlerini yaptıktan sonra model nesnesi tanımlama aşamasına geçtim. Karar ağacı algoritması kullanmak için bir model nesnesi tanımlayalım.

```
# Karar ağacı algoritması ile bir model nesnesi tanımlayalım.
model = DecisionTreeClassifier()
```

Model nesnemizi tanımladıktan sonra tıpkı Gaussian Naive Bayes algoritmasında olduğu gibi modelimi eğitmem gerekiyor. Modelimi eğitmek için şu kodu yazdım:

```
# Oluşturduğumuz modeli verilerimiz ile eğitelim.
model.fit(X_train, y_train)
```

Modelimizi eğittikten ve istenen doğruluk değerine ulaştıktan sonra tıpkı Gaussian Naive Bayes algoritmasında yaptığımız gibi modelimizi joblib kütüphanesi yardımıyla dosyaya kaydederek daha sonrasında dosyadan okuyarak modelimizi kullanabiliriz. *Decision Tree Classifier* algoritması ile oluşturduğumuz modelimizi joblib kütüphanesi ile dosyaya kaydetmek için aşağıdaki kodu yazdım.

```
joblib.dump(model, 'egitilmis_model_DecisionTree.joblib')
```

Oluşturduğumuz model dosyasını kullanmak için aşağıdaki kodu yazdım.

```
model = joblib.load('egitilmis_model_DecisionTree.joblib')
```

Oluşturduğumuz model dosyasını kullanmak istediğimizde `model.fit(X_train, y_train)` satırını kapatmalı veya silmeliyim ki modelim tekrar tekrar eğitilmesin.

Modelimi eğittikten sonra artık test veri setim ile tahminde bulunma işlemlerini yapma aşamasına geçtim. Bunun için aşağıdaki kod satırını yazdım.

```
y_pred = model.predict(X_test)
```

Tahmin Sonuçlarını Test Sonuçları ile Karşılaştırma

Hata matrisi (confusion matrix) kullanarak modelin başarısını ölçelim:

```
print('Karmaşıklık Matrisi:')
print(irisData.target_names)
print(confusion_matrix(y_test, y_pred))

Karmaşıklık Matrisi:
['setosa' 'versicolor' 'virginica']
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
```

Başarı Oranının Hesaplanması

Başarı oranını (accuracy_score) hesaplayarak ekrana yazdıralım.

```
# Başarı Oranı
# test_size = 0.25 ile DecisionTreeClassifier algoritmasını kullanarak %97 oranında başarı ile çalıştı.
print ("Doğruluk (%) : ", 100 * accuracy_score(y_test, y_pred))
```

Oluşturduğumuz modeli görsel olarak incelemek için ağaç nesnesi ile export ediyorum. dot uzantılı dosyayı Visual Studio Code ile açabiliriz. (Ağacı görüntülemek için joaompinto.vscode-graphviz eklentisi yüklü olmalısı tavsiye ederim.)

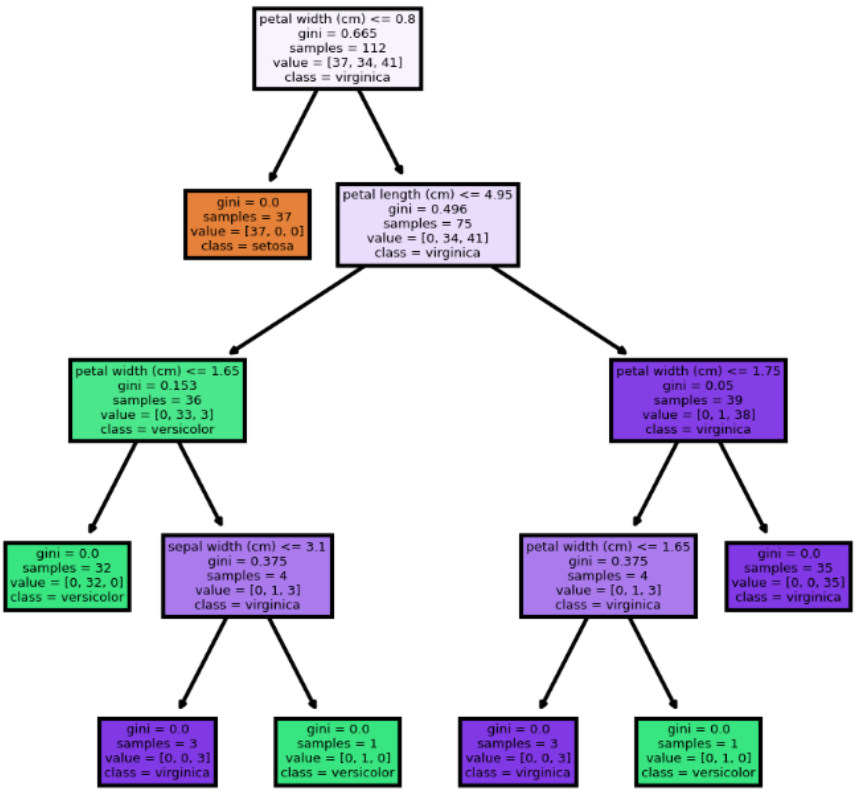
```
tree.export_graphviz(
    model,
    out_file = 'model_tree.dot',
    label = 'all',
    filled = True,
    rounded = True)
```

Karar ağacını oluşturmak ve dışarı aktarmak için tree kütüphanesinin `export_graphviz` fonksiyonunu kullandık. Ayrıca karar ağacını dot uzantılı dosyanın yanı sıra resim olarak da kaydettim. Bunun için aşağıdaki kod satırlarını yazdım.

```
# modeli görsel olarak ekranda gösterelim ve resim olarak kaydedelim.
fn=['sepal length (cm)','sepal width (cm)','petal length (cm)','petal width (cm)']
cn=['setosa', 'versicolor', 'virginica']
fig, axes = plt.subplots(nrows = 1, ncols = 1, figsize = (4,4), dpi=300)
tree.plot_tree(model, feature_names = fn, class_names=cn, filled = True);
fig.savefig('model_tree.png')
```

Yukarıdaki kod satırını çalıştırdıktan sonra oluşan karar ağacım dot ve jpg uzantılı dosyalar olarak kaydedilmiş oldu. Oluşan karar ağacım şöyledir.

Şekil 3. Karar Ağacı



Yorum:

Matrisin köşegenlerin toplamı bize test edilen kayıt sayısını vermektedir. Karmaşıklık matrisinin köşegenleri toplamı 38'dir. Bu noktadan hareketle karmaşık matrisini yorumlayalım: 38 tane test kaydında Setosa sınıfına ait 13 tane kayıt varmış ve hepsi doğru tahmin edilmiş. 16 tane versicolor varmış bunların 15 tanesini doğru 1 tanesini ise virginica olarak tahmin edilmiş. 9 tane virginica varmış ve hepsi de başarıyla virginica olarak tahmin edilmiş.

Şekil 4. Karmaşıklık Matrisi

```
Karmaşıklık Matrisi:
['setosa' 'versicolor' 'virginica']
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
```

Başarı oranını yorumlayacak olursak, projemizde Iris veri setinde `test_size = 0.25` ile *Decision Tree Classifier* sınıfını kullanarak modelimizi eğittik sonrasında ise tahminlerde bulunmasını istedik. Bu tahminler sonucunda başarı oranı % 97 olarak bulundu. Bu skor algoritmamızın başarılı bir şekilde tahminde bulunduğunu göstermektedir.

Yapılan görüntü işleme uygulamasında hazır veri seti kullanılmıştır. Bu nedenle bu veri seti içerisinden test verisi oluşturularak model test edilmiştir. Test için ayrılan oran değiştikçe farklı doğruluk oranları almak mümkündür.

2. Python dili, "Tensor-Flow" kütüphanesi ile Derin Öğrenme (görüntü işlemeye yönelik) uygulaması gerçekleştiriniz. Mevcut kod üzerinde satır satır değişiklikler yaparak farklı algoritma ve parametrelerle görüntü tanıma sistemi geliştirebilirsiniz.

Tanımlar

Derin öğrenme (aynı zamanda derin yapılandırılmış öğrenme, hiyerarşik öğrenme ya da derin makine öğrenmesi) bir veya daha fazla gizli katman içeren yapay sinir ağları ve benzeri makine öğrenme algoritmalarını kapsayan çalışma alanıdır. Yani en az bir adet yapay sinir ağının (YSA) kullanıldığı ve birçok algoritma ile, bilgisayarın eldeki verilerden yeni veriler elde etmesidir. Derin öğrenme gözetimli, yarı gözetimli veya gözetimsiz olarak gerçekleştirilebilir. Derin yapay sinir ağları pekiştirmeli öğrenme yaklaşımıyla da başarılı sonuçlar vermiştir (Vikipedi, 2021).

Görüntü işleme isim (Almanca Bildbearbeitung) ölçülmüş veya kaydedilmiş olan elektronik (dijital) görüntü verilerini, elektronik ortamda (bilgisayar ve yazılımlar yardımı ile) amaca uygun şekilde değiştirmeye yönelik yapılan bilgisayar çalışmasıdır. Görüntü işleme, daha çok, kaydedilmiş olan, mevcut görüntüleri işlemek, yani mevcut resim ve grafikleri, değiştirmek, yabancılaştırmak ya da iyileştirmek için kullanılır (Vikipedi, 2021a).

TensorFlow, bir dizi görev arasında veri akışı ve türevlenebilir programlama için kullanılan ücretsiz ve açık kaynaklı bir yazılım kütüphanesidir. Sembolik bir matematik kütüphanesidir ve sinir ağları gibi makine öğrenimi uygulamaları için de kullanılır. Google'da hem araştırma hem de üretim için kullanılır. TensorFlow, Google Brain ekibi tarafından dahili Google kullanımı için geliştirilmiştir. 9 Kasım 2015'te Apache Lisansı 2.0 altında yayınlandı (Vikipedi, 2021b).

Uygulama

İkinci soruyu gerçekleştirmek için öncelikle bilgisayarımızda Python kurulu olmalıdır. Editör olarak Visual Studio Code, PyCharm veya Jupyter Notebook kullanılabilir. Kullanım kolaylığı açısından ben Jupyter Notebook kullanmayı tercih ettim. Editörü açtıktan sonra ilk yapmamız gereken ilgili kütüphaneleri projemize dahil etmek olacaktır. Bunun için aşağıdaki kodları yazdım.

```
# Proje ile ilgili kütüphaneler dahil ediliyor.
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```

TensorFlow kütüphanesi ile görüntü işlemek için öncelikle bir veri setine ihtiyacımız var. Ben Zalando Araştırma Ekibi tarafından geliştirilen ve kıyafetlerden oluşan bir veri seti olan Fashion-MNIST veri setini kullanacağım. Bu veri seti içerisinde 60.000 resim modeli eğitmek için, 10.000 resim ise eğitilen modelin performansını ölçmek için kullanılıyor. Veri seti ilgili bilgileri öğrenmek adına aşağıdaki kodu çalıştırdım.

```
# veri sayılarını getir.
# eğitim setinde her bir görüntünün 28 x 28 piksel olarak temsil edildiği 60.000 görüntü bulunmaktadır.
train_images.shape
(60000, 28, 28)

# test veri seti bilgilerini getir.
# test veri setinde 10000 adet 28x28 piksel ebatlarında görüntü bulunmaktadır.
test_images.shape
(10000, 28, 28)
```

Fashion-MNIST veri setini uygulamama dahil etmek için aşağıdaki kodları yazdım.

```
#Fashion MNIST veri kümesini içe aktar
fashion_mnist = tf.keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

Veri setini uygulamamıza yükledikten sonra veri etiketleri ile eşleştirdim ve daha sonra görüntüleri ekrana çizerken kullanmak amacıyla `class_names` isimli değişkene atadım.

```
#Her görüntü tek bir etiketle eşleştirilir.
#Sınıf adları veri kümesine dahil edilmediğinden, daha sonra görüntüleri
#çizerken kullanmak için class_names değişkenine atadım.
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

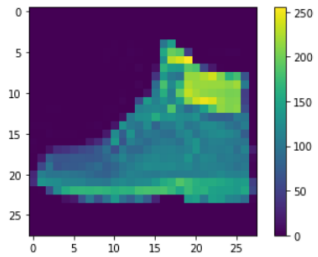
Ağı eğitmeden önce veriler önceden işlenmelidir. Eğitim setindeki 90. resim incelendiğinde, piksel değerlerinin 0 ile 255 aralığında olduğunu görülmektedir. Bu değerleri sinir ağı modeline beslemeden önce 0 ila 1 aralığında ölçeklendirilmelidir. Bu işlemi yapmak için değerlerini 255'e bölmeliyiz. Eğitim seti ve test setinin aynı şekilde ön işlemden geçirilmesi önemlidir. İşlemleri gerçekleştirmek için yazdığım kod aşağıdadır.

```
# Ağı eğitmeden önce veriler önceden işlenmelidir.
# Eğitim setindeki 90. resmi incelerseniz, piksel değerlerinin
# 0 ile 255 aralığında olduğunu göreceksiniz:
plt.figure()
plt.imshow(train_images[90])
plt.colorbar()
plt.grid(False)
plt.show()

# Bu değerleri sinir ağı modeline beslemeden önce 0 ila 1 aralığında ölçeklendirilir.
# Bunu yapmak için piksel değerlerini 255'e bölmeliyiz.
# Eğitim seti ve test setinin aynı şekilde ön işlemden geçirilmesi önemlidir:

train_images = train_images / 255.0
test_images = test_images / 255.0
```

Şekil 5. Veri setinin 90. resmi



```
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()
```

Verilerin doğru biçimde olduğunu ve ağı kurmaya ve eğitmeye hazır olduğunu doğrulamak için eğitim setinden ilk 25 görüntüyü gösterelim ve her görüntünün altında sınıf adını yazalım. Aşağıdaki şekillerde ilgili kod ve ilk 25 veri görülmektedir. Ekrana görselleri basmak için `matplotlib.pyplot` kullanılmıştır. Yandaki şekilde görüldüğü üzere verilerimiz doğru ve artık modeli oluşturup eğitmeye hazırdır. Şimdi modelimizi/ağımızı oluşturalım. Sinir ağını oluşturmak modelin katmanlarını yapılandırmayı ve ardından modeli derlemeyi gerektirmektedir. Öncelikle ağı oluşturalım. TensorFlow bir model tanımlamanın iki yolunu sunar: Sequential, katmanları yığmak için kullanılır ve en sık kullanılan model budur.

Modellerin katmanlardan oluşmaktadır, katmanlar sinir ağını oluşturan temel bloklar olarak düşünülebilir. İnsan sinir hücrelerinden esinlenilerek gerçekleştirilmeye çalışılan bu yapıda, katmanlar girdi verilerini işleyerek bir çıkış üretirler ve bu çıkışlar ise bir başka katmanın girdisi olabilir. TensorFlow'un içerdiği temel(çekirdek) katman çeşitlerinden bazıları; Dense layer : girdideki her bir düğüm çıkıştaki her bir düğüm ile bağlıdır. Flatten layer : Matris formundaki veriyi düzleştirmek için kullanılır (Delibaşoğlu, 2017).

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)
])
```

Yukarıdaki kod satırları ile bir model oluşturduk. Oluşturduğumuz modeli derleyelim. Bunun için aşağıdaki kod satırını yazdım.

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

Modelimi oluşturup derledikten sonra modelimi eğitme aşamasına geldim. Başlangıçta train_images ve test_images olarak parçaladığım veri setimden train_images değişkenine attığım veriler ile modelimi eğiteceğim. Bunun için ilgili kod ve modelin eğitim aşamaları aşağıdadır.

```
# model eğitiliyor
model.fit(train_images, train_labels, epochs=10)
```

Şekil 6. Modelin Eğitim Aşamaları

```
Epoch 1/10
1875/1875 [==============================] - 1s 518us/step - loss: 0.4970 - accuracy: 0.8257
Epoch 2/10
1875/1875 [==============================] - 1s 513us/step - loss: 0.3719 - accuracy: 0.8645
Epoch 3/10
1875/1875 [==============================] - 1s 512us/step - loss: 0.3327 - accuracy: 0.8790
Epoch 4/10
1875/1875 [==============================] - 1s 521us/step - loss: 0.3104 - accuracy: 0.8853
Epoch 5/10
1875/1875 [==============================] - 1s 518us/step - loss: 0.2923 - accuracy: 0.8923
Epoch 6/10
1875/1875 [==============================] - 1s 518us/step - loss: 0.2771 - accuracy: 0.8978
Epoch 7/10
1875/1875 [==============================] - 1s 521us/step - loss: 0.2665 - accuracy: 0.9007
Epoch 8/10
1875/1875 [==============================] - 1s 524us/step - loss: 0.2561 - accuracy: 0.9054
Epoch 9/10
1875/1875 [==============================] - 1s 526us/step - loss: 0.2462 - accuracy: 0.9071
Epoch 10/10
1875/1875 [==============================] - 1s 529us/step - loss: 0.2379 - accuracy: 0.9099
```

Model eğitilirken, kayıp ve doğruluk ölçümleri görüntülenir. Bu model, eğitim verilerinde yaklaşık 0,90 (veya %90) doğruluğa ulaşmıştır.

`model_fit` fonksiyonuna 3 adet parametre gönderdik. Bunlar `train_images, train_label` ve `epochs` parametreleridir. `train_images` eğitim verilerim, train_label verilere ait etiketler ve `epoch` ise, model eğitilirken verilerin modelden kaç kez geçiş yapacağını belirtmektedir. Ardından modelin test veri kümesinde nasıl performans gösterdiğini bulmak için modelimi evaulate ederek Test accuarcy değerini ekrana bastırdım. Bu değer %88 çıktı test veri setimizi %88 başarı ile çalıştığını göstermektedir.

```
# Doğruluğu değerlendir
# modelin eğitilmesinin ardından, modelin test veri kümesinde nasıl performans gösterdiğini karşılaştır
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
print('\nTest accuracy: (%)', 100*test_acc)
```

Her şey doğru gitti modelimizi eğittik ve artık test setimiz üzerinde tahminler ürettirmeye hazırız. Modelimize yeni bir katman ekliyoruz. Böylece yorumlanmasını kolaylaştıracağız. Ardından test setindeki her bir görüntü için etiketi tahmin edeceğiz.

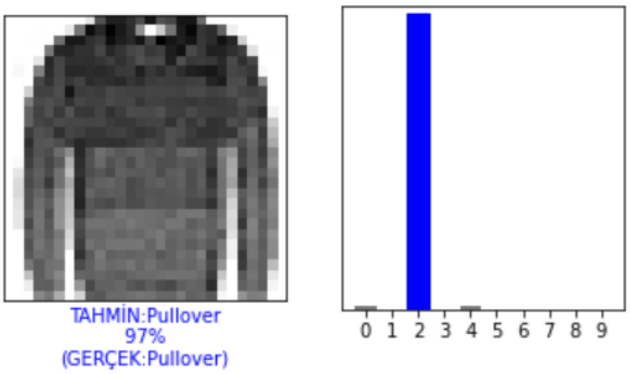
```
# Eğitilen modeli bazı görüntüler hakkında tahminlerde bulunalım
# Sonuçları yorumlanması daha kolay olan olasılıklara dönüştürmek için bir softmax katmanı eklemeliyiz.
probability_model = tf.keras.Sequential([model, tf.keras.layers.Softmax()])
```

Tahmin işlemlerinden sonra tahminleri doğrulamamız kesin sonuç almak ve modelimizin doğru çalışmasını kontrol etmek açısından önemlidir. Test seti kullanılarak eğitilen model ile bazı görüntüleri (indis numarasını verdiğim görüntüyü) tahmin etmesini istiyorum. Bunun için i değerini değiştirerek istenilen görüntü ile ilgili tahminde bulunabilmesini istiyorum.

```
#Burada model, test setindeki her bir görüntü için etiketi tahmin etmiştir.
predictions = probability_model.predict(test_images)
```

```
i = 20
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()
```

Yukarıda verilen kodda # i değeri 20 girilirse %100 doğrulukta tahmin yapılan bir örnek görülebilir.

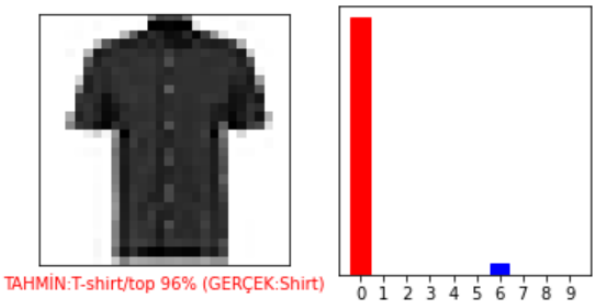



```
print('SONUÇLAR')
print('Tahmin edilen görüntü: ' + class_names[np.argmax(predictions[i])])
print('Gerçek görüntü: ' + class_names[np.argmax(predictions[i])])
print('Doğruluk Oranı: ', 100*np.max(predictions[i]))

SONUÇLAR
Tahmin edilen görüntü: Pullover
Gerçek görüntü: Pullover
Doğruluk Oranı:  98.2964038848877
```

Yukarıdaki sonuçları yorumlayacak olursak modelimiz %97 başarı oranıyla indis numarası verilen görüntünün Pullover olduğundan emin ve kontrol ettiğimizde bu tahminin doğru olduğunu görmekteyiz.

Eğer i değeri 40 girilirse %96 doğrulukta tahmin yapılan bir örnek görülebilir.



Yukarıdaki sonuçları yorumlayacak olursak modelimiz %96 başarı oranıyla indis numarası verilen görüntünün T-shirt/top olduğundan emin ancak kontrol ettiğimizde bu tahminin doğru olmadığını, gerçekte bu görüntünün Shirt olduğunu görmekteyiz.

Tahmin sonuçlarını görüntü ve grafik olarak verilmesi için yazdığım kod aşağıdadır.

```
# Resmi çizmek için kullanılan fonksiyon
def plot_image(i, predictions_array, true_label, img):
  true_label, img = true_label[i], img[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  plt.imshow(img, cmap=plt.cm.binary)

  predicted_label = np.argmax(predictions_array)
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'

  plt.xlabel("TAHMİN:{} {:2.0f}% (GERÇEK:{})".format(class_names[predicted_label],
                                100*np.max(predictions_array),
                                class_names[true_label]),
                                color=color)

# grafiğin çizilmesi için kullanılan fonksiyon
def plot_value_array(i, predictions_array, true_label):
  true_label = true_label[i]
  plt.grid(False)
  plt.xticks(range(10))
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)

  thisplot[predicted_label].set_color('red')
  thisplot[true_label].set_color('blue')
```

Yapılan görüntü işleme uygulamasında hazır veri seti kullanılmıştır. Bu nedenle bu veri seti içerisinden test verisi oluşturularak model test edilmiştir. Test için ayrılan oran değiştikçe farklı doğruluk oranları almak mümkündür.

KAYNAKÇA

Delibaşoğlu İ. (2017). *Keras ile Derin Öğrenmeye Giriş-Sınıflama Örneği*. http://ibrahimdelibasoglu.blogspot.com/2017/09/keras-ile-derin-ogrenmeye-giris-snflama.html adresinden 27 Mayıs 2021 tarihinde erişildi.

Karabay, A. (2021). Scikit-learn Nedir? https://www.karabayyazilim.com/blog/python/scikit-learn-nedir-2020-02-12-062241 adresinden 24 Mayıs 2021 tarihinde erişilmiştir.

Navlani A. (2018). *Naive Bayes Classification using Scikit-learn*. https://www.datacamp.com/community/tutorials/naive-bayes-scikit-learn adresinden 24 Mayıs 2021 tarihinde erişildi.

Scikit-Learn (2021). *User Guide*. https://scikit-learn.org/stable/user_guide.html adresinden 24 Mayıs 2021 tarihinde erişildi.

Şirin, E. (2021). *Iris Verisi ile Sınıflandırma Alıştırması (Python Scikit-Learn).* https://www.veribilimiokulu.com/iris-verisi-ile-siniflandirma-alistirmasi-python-scikit-learn/ adresinden 24 Mayıs 2021 tarihinde erişilmiştir.

Veri Bilimi (2021). *Scikit-Learn'e Hızlı Başlangıç.* https://veribilimcisi.com/2017/07/13/scikit-learne-hizli-baslangic/#scikitlearnilemakineogrenmesinegiris adresinden 24 Mayıs 2021 tarihinde erişildi.

Vikipedi (2021). *Derin Öğrenme*. https://tr.wikipedia.org/wiki/Derin_Öğrenme adresinden 27 Mayıs 2021 tarihinde erişildi.

Vikipedi (2021a). *Görüntü İşleme*. https://tr.wikipedia.org/wiki/Görüntü_işleme adresinden 27 Mayıs 2021 tarihinde erişildi.

Vikipedi (2021b). *Görüntü İşleme*. https://tr.wikipedia.org/wiki/TensorFlow adresinden 27 Mayıs 2021 tarihinde erişildi.

Wikipedia (2021). *Scikit-Learn*. https://en.wikipedia.org/wiki/Scikit-learn adresinden 24 Mayıs 2021 tarihinde erişildi.

Zobu İ. (2019). *Naive Bayes: Teorisi ve Python uygulaması*. https://medium.com/kaveai/naive-bayes-ve-uygulamaları-d7d5a56c689b adresinden 24 Mayıs 2021 tarihinde erişildi.