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
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.)

```
model = GaussianNB()
```

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)
```

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

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

Başarı Oranının Hesaplanması

```
# Başarı Oranı
# Bu örneğimizde modelimiz 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.

Decısıon 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.

```
model = DecisionTreeClassifier()
```

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

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

```
Doğruluk (%) :  97.36842105263158
```

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.)

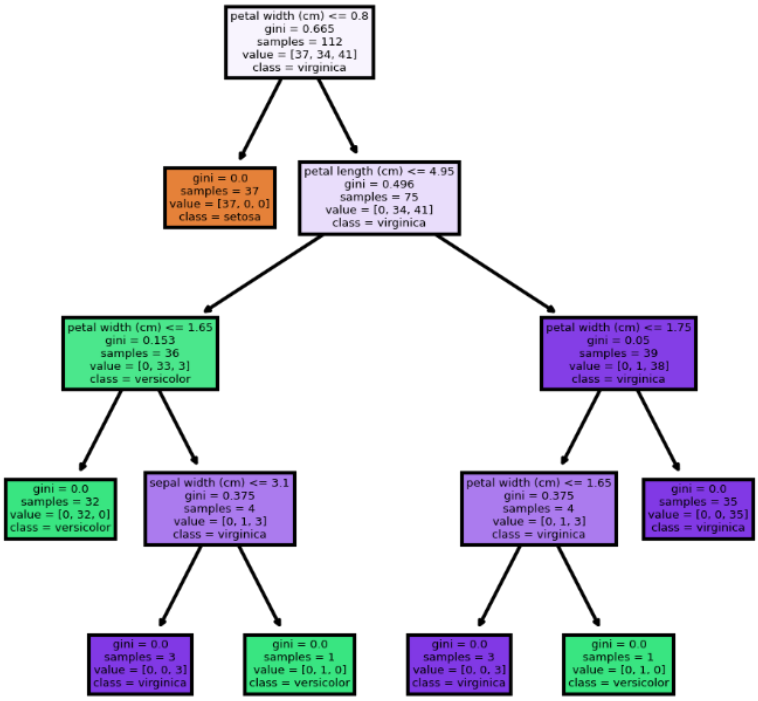
```
tree.export_graphviz(model, out_file = 'model_tree.dot', label = 'all', filled = True, rounded = True)
```

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('imagename.png')
```

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

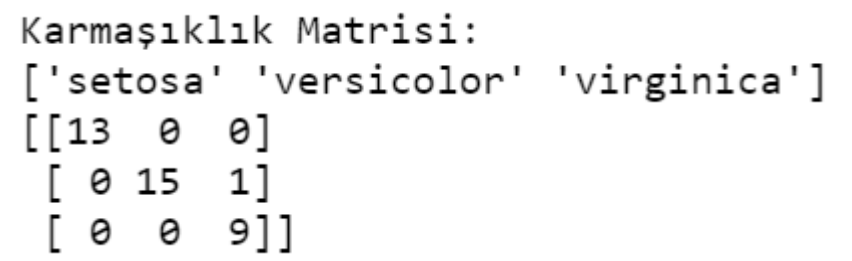
Şekil 3. Karar Ağacı



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



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. Eğer `test_size` değişkeninin değeri değiştirilerek program yeniden çalıştırılır ise farklı sonuçlar elde edilebilir.

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.

KAYNAKÇA

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.

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-uygulamalar%C4%B1-d7d5a56c689b adresinden 24 Mayıs 2021 tarihinde erişildi.