

Pratikte BT & BS Uygulamaları

Makine Öğrenmesi ve Derin Öğrenme

20.05.2021

—

Onur Osman Güle

G171210021 - 2A

onur.gule@ogr.sakarya.edu.tr

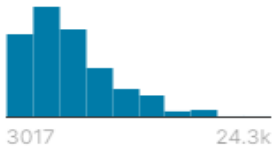
ScikitLearn ile Makine Öğrenmesi Uygulaması

Uygulama Hakkında

Uygulama olarak makine öğrenmesi ile yazılım geliştirme departmanında çalışanların maaşlarını tahmin etmeye dayalı bir uygulama yapmayı planladım.

< **calisan_data.csv** (269.51 KB)

Detail Compact Column

▲ Departman	▲ Şehir	▲ DoğumTarihi	# Maas
Yazılım Geliştirme 60%	İstanbul 48%	2141 unique values	 3017 24.3k
Müşteri İlişkileri ve ... 15%	Ankara 27%		
Other (614) 25%	Other (611) 25%		
Yazılım Geliştirme	Ankara	02 Kasım 1979	11679
Yazılım Geliştirme	Bursa	03 Şubat 1987	5206
Yazılım Geliştirme	İstanbul	22 Ekim 1987	8670
Yazılım Geliştirme	İstanbul	11 Haziran 1987	5875
Yazılım Geliştirme	İstanbul	25 Ekim 1992	5828
Müşteri İlişkileri ve Satış	Bursa	13 Ağustos 1976	13114
Yazılım Geliştirme	Kocaeli	21 Şubat 1981	9273
Yazılım Geliştirme	İstanbul	17 Mart 1996	5361

Öncelikle isteğime uygun bir veri seti buldum. Kurgusal bir şirketin departmanları, doğum tarihleri ve maaşlarını içeriyor bu veri seti. Gidiş yolum departman ve doğum tarihi ile birlikte tecrübeyi hesaplayıp maaşı tecrübeye göre Naive Bayes ve Karar Ağaçları algoritmalarını kullanarak tahmin etmekte.



Uygulamaya Giriş

Öncelikle pandas ile veriyi çekip gerekli değişkenleri tanımladım.

```
import pandas as pd
data = pd.read_csv("calisan_data.csv")
data = data[['Departman', 'DogumTarihi', 'Maas']]
dataset_year = 2019 #dataset tarihi
tecrube_baslangic_yasi = 21 #21 yaşından beri aynı sektörde çalıştığı farzediliyor.
```

İşimize yaramayan kolonları temizledim, veri setinin oluşturulduğu tarihi ve tecrübe başlangıç yaşı adında bir değişken tanımladım. Üniversiteden 21 yaşında mezun olduğunu düşünerek tecrübeyi doğum tarihi ile hesaplayacağım.

```
for index, row in data.iterrows():
    data['DogumTarihi'][index] = (row['DogumTarihi'].split(" ")[2])
#datasetten doğum ay ve günü silindi.
for index, row in data.iterrows():
    data['Maas'][index] = int(row['Maas']) - int(row['Maas']%1000)
#Integer dönüşümü
data['MaxTecrube'] = pd.Series()
for index, row in data.iterrows():
    data['MaxTecrube'][index] = (dataset_year - tecrube_baslangic_yasi - int(row['DogumTarihi']))
#Varsayımsal olarak max tecrübe oluşturduk.
data = data[['Departman', 'MaxTecrube', 'Maas']]
data
```

Doğum tarihinde yalnızca yıl kalacak şekilde düzenledim, maaşları 1000'lere yuvarladım, MaxTecrube adında bir kolon oluşturup örneğin 1996'da doğan birinin tecrübe yaşı = 2019 - 21 - 1996 = 2 yıl olacak şekilde güncelledim ve son olarak doğum tarihi ile işim kalmadığından ilgili kolonu ayrıştırdım.



	Departman	MaxTecrube	Maas
0	Müşteri İlişkileri ve Satış	13.0	9000
1	Yazılım Geliştirme	19.0	11000
2	Yazılım Geliştirme	19.0	11000
3	Yazılım Geliştirme	11.0	5000
4	Yazılım Geliştirme	11.0	8000
...
2440	Yazılım Geliştirme	23.0	12000
2441	Yazılım Geliştirme	12.0	6000
2442	Yazılım Geliştirme	15.0	8000
2443	Yazılım Geliştirme	14.0	9000
2444	Yazılım Geliştirme	20.0	14000

2445 rows × 3 columns

Sonuç olarak veri setim yukarıdaki şekilde güncellendi.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data['Departman'] = le.fit_transform(data['Departman'])
```

Sonrasında Departman kolonu string veri barındırdığından bunu gruplayıp hesaplanabilir veriye dönüştürmek için LabelEncoder kullandım.

	Departman	MaxTecrube	Maas
0	1	13.0	9000
1	2	19.0	11000
2	2	19.0	11000
3	2	11.0	5000
4	2	11.0	8000
...
2440	2	23.0	12000
2441	2	12.0	6000
2442	2	15.0	8000
2443	2	14.0	9000
2444	2	20.0	14000

2445 rows × 3 columns

Son olarak yukarıda görüldüğü gibi veri setim işlem yapılabilecek ölçüye gelmişti.



Model Oluşturma

Tahmin etmek için öncelikle verilerime dayalı bir model oluşturmam gerekiyor.

```
from sklearn.model_selection import train_test_split
training_set, validation_set = train_test_split(data, test_size = 0.2, random_state = 21)
X_train = training_set.iloc[:,0:-1].values
Y_train = training_set.iloc[:, -1].values
X_val = validation_set.iloc[:,0:-1].values
y_val = validation_set.iloc[:, -1].values
```

Öncelikle veriyi eğitim ve test olmak üzere %80, %20 şeklinde ayırdım. Sonrasında X alanına Departman ve MaxTecrube, Y alanına ise Maas sütunlarını tanımladım.

```
:
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, Y_train)
```

```
, GaussianNB()
```

Sonrasında sklearn kütüphanesinden naive_bayes'deki GaussianNB'yi kullanarak sınıflandırma algoritması oluşturdum.

```
:
from sklearn.tree import DecisionTreeClassifier
tclassifier = DecisionTreeClassifier()
tclassifier.fit(X_train, Y_train)
```

Ardından Sklearn'deki DecisionTree yani Karar Ağaçları da kullanarak ayrıca bir sınıflandırma algoritması oluşturdum.

```
:
def accuracy(confusion_matrix):
    carpraz_top = confusion_matrix.trace()
    el_top = confusion_matrix.sum()
    return float(carpraz_top) / float(el_top)
```

Sonrasında verimi ölçmek amacıyla accuracy adında bir fonksiyon oluşturdum.



Tahmin Etme

Naive Bayes Algoritması Verimi

```
: y_pred = classifier.predict(X_val)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_val, y_pred)

print("VERIM = ", accuracy(cm))
```

VERIM = 0.25357873210633947

Karar Ağaçları Algoritması Verimi

```
y_pred = tclassifier.predict(X_val)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_val, y_pred)

print("VERIM = ", accuracy(cm))
```

VERIM = 0.22699386503067484

Sonuç olarak iki alitritmadan da verim pek istediğimiz gibi gelmedi, verilerin kurgusal olması ve az veri olmasından kaynaklı olarak verimimiz naive bayes algoritmasında %25, karar ağaçları algoritmasında %22 geldi. Bu tahminlerimizi daha düşük doğru orantılı yapacaktır.



Tahminlerimizde aşağıdaki gibi farklı parametrelerle farklı sonuçlar çıkmıştır.

```
:
Ornek = [
    [2,2], #2 yıllık tecrübesi olan yazılım geliştiricisi
    [2,6], #6 yıllık tecrübesi olan yazılım geliştiricisi
    [1,5], #5 yıllık tecrübesi olan müşteri ilişkileri çalışanı
    [2,19], #19 yıllık tecrübesi olan yazılım geliştiricisi
]
tahminler = classifier.predict(Ornek)
tahminler

array([ 3000,  5000,  4000, 11000])
```

Naive Bayes algoritmasında:

2 yıllık tecrübesi olan yazılım geliştiricisi 3.000TL maaş alırken 6 yıllık tecrübesi olan yazılım geliştiricisi 5.000TL maaş almaktadır. 5 yıllık müşteri ilişkileri çalışanı ise 4.000TL almaktadır. 19 yıllık senior bir yazılım geliştiricisi ise 11.000TL almaktadır.

```
Ornek = [
    [2,2], #2 yıllık tecrübesi olan yazılım geliştiricisi
    [2,6], #6 yıllık tecrübesi olan yazılım geliştiricisi
    [1,5], #5 yıllık tecrübesi olan müşteri ilişkileri çalışanı
    [2,19], #19 yıllık tecrübesi olan yazılım geliştiricisi
]
tahminler = tclassifier.predict(Ornek)
tahminler

array([5000, 7000, 3000, 8000])
```

Karar Ağaçları algoritmasında:

2 yıllık tecrübesi olan yazılım geliştiricisi 5.000TL maaş alırken 6 yıllık tecrübesi olan yazılım geliştiricisi 7.000TL maaş almaktadır. 5 yıllık müşteri ilişkileri çalışanı ise 3.000TL almaktadır. 19 yıllık senior bir yazılım geliştiricisi ise 8.000TL almaktadır.

Sonuç olarak veriler çok tutarlı olmasa da sonuçlar çok da yanıltıcı değildi. Naive Bayes ile daha gerçekçi sonuçlar yakaladık. Daha fazla veri olsaydı daha da gerçekçi sonuçlar yakalayabilirdik. ScikitLearn ve Naive Bayes ile birlikte bir tahmin etme uygulaması oluşturduk.

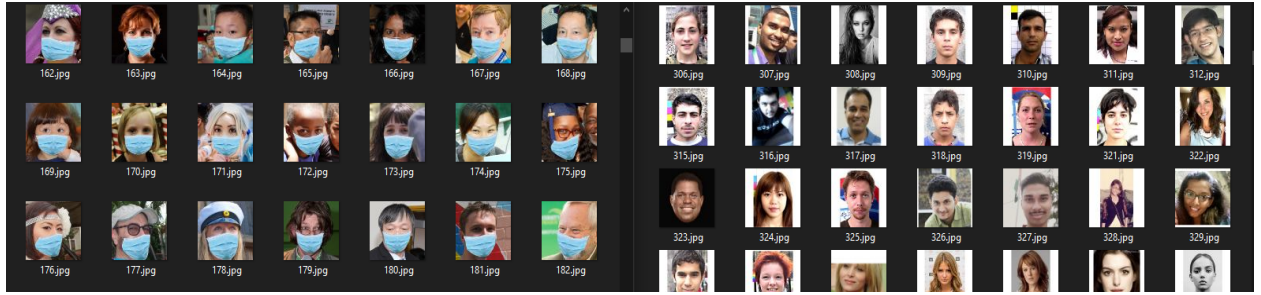
Github: <https://github.com/onurgule/MaasHesabi>



TensorFlow ile Derin Öğrenme

Uygulama Hakkında

Maske takılmış mı, takılmamış mı, doğru takılmış mı diye kontrol eden bir uygulama hazırlamayı planladım. İnternette maskeli ve maskesiz kişileri içeren bir veri seti buldum.



Yukarıda görüldüğü gibi sol taraftaki klasörde maskeli insanlar, sağ taraftaki klasörde ise maskesiz veya maskesini yanlış kullanan insanlar mevcut.

Bu verileri tensorflow ile eğitip bir model oluşturduktan sonra kendi fotoğraflarım üzerinde test edeceğim.

Uygulamaya Giriş

Öncelikle model eğiteceğim için güçlü bir bilgisayar ihtiyacım olduğundan Google Colab kullanmayı tercih ettim.

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dense, Flatten, BatchNormalization, Conv2D, MaxPool2D, Dropout
from tensorflow.keras.optimizers import Adam
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
from google.colab import files
```

Gerekli pandas, numpy, tensorflow, keras, google colab kütüphanelerini ekledim. Burada google colab benim veri setimi ve kendi fotoğraflarımı çekmem için kullanılacak, lokaldeki bir jupyter notebook için gerekli değil.




```
[10] batch_size = 8
    epochs = 30

    #örnek resimleri eğitim ve test olarak ayırıştırıyoruz.
    directory = 'ornek_resimler'
    train_datagen = ImageDataGenerator(validation_split=0.2,
                                       rescale = 1./255,
                                       rotation_range=40,
                                       width_shift_range=0.2,
                                       height_shift_range = 0.2,
                                       zoom_range = 0.2,
                                       horizontal_flip=True,
                                       fill_mode='nearest')

    train_generator = train_datagen.flow_from_directory(
        directory,
        target_size = (70, 70),
        batch_size = batch_size,
        color_mode="rgb",
        class_mode = 'binary',
        seed=2020,
        subset = 'training')

    validation_generator = train_datagen.flow_from_directory(
        directory,
        target_size = (70, 70),
        batch_size = batch_size,
        color_mode="rgb",
        class_mode = 'binary',
        subset = 'validation')
```

İnternette aldığım veri seti içerisindeki örnek resimleri train ve test olarak ayırıştırıyorum.

```
imgs, labels = next(train_generator)
def plotImages(images_arr):
    fig, axes = plt.subplots(1, batch_size, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip( images_arr, axes):
        ax.imshow(img)
        ax.axis('off')
    plt.tight_layout()
    plt.show()

#0: maskeli, 1:maskesiz şekilde plotluyoruz.
plotImages(imgs);
print(labels);
```

Fotoğrafları 0 sorunsuz, 1 sorunlu şekilde kullanılan maske olarak ayırıştırıyorum.



Model Oluşturma

Veri setinden kendi fotoğraflarıma uygulamak için bir model oluşturmam gerekiyor.

```
#modelimizi oluşturalım.
model = Sequential([
    Conv2D(filters=32, kernel_size=(3,3),activation='relu',padding='same',input_shape=(70,70,3)),
    MaxPool2D(pool_size=(2,2), strides=2),
    Conv2D(filters=32, kernel_size=(3,3), activation='relu', padding='same'),
    MaxPool2D(pool_size=(2,2), strides=2),
    Conv2D(filters=64, kernel_size=(3,3), activation='relu', padding='same'),
    MaxPool2D(pool_size=(2,2), strides=2),
    Flatten(),
    Dense(units=64, activation='relu'),
    Dense(units=1, activation='sigmoid'),
])

model.summary()
model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])
history = model.fit(train_generator, epochs = epochs, validation_data= validation_generator)
#modelimizi eğitiyoruz.
```

Adam algoritmasını kullanarak modelimizi eğitiyoruz. 30 epoch ile çalışacaktır.

```
Epoch 1/30
1002/1002 [=====] - 76s 75ms/step - loss: 0.5384 - accuracy: 0.7078 - val_loss: 0.4329 - val_accuracy: 0.8067
Epoch 2/30
1002/1002 [=====] - 76s 75ms/step - loss: 0.3116 - accuracy: 0.8699 - val_loss: 0.3321 - val_accuracy: 0.8546
Epoch 3/30
1002/1002 [=====] - 76s 76ms/step - loss: 0.2570 - accuracy: 0.9001 - val_loss: 0.2703 - val_accuracy: 0.8916
Epoch 4/30
1002/1002 [=====] - 79s 79ms/step - loss: 0.2323 - accuracy: 0.9070 - val_loss: 0.3182 - val_accuracy: 0.8661
Epoch 5/30
1002/1002 [=====] - 77s 77ms/step - loss: 0.2068 - accuracy: 0.9214 - val_loss: 0.2364 - val_accuracy: 0.9021
Epoch 6/30
1002/1002 [=====] - 78s 77ms/step - loss: 0.1906 - accuracy: 0.9283 - val_loss: 0.2643 - val_accuracy: 0.9006
Epoch 7/30
1002/1002 [=====] - 78s 78ms/step - loss: 0.1673 - accuracy: 0.9369 - val_loss: 0.2377 - val_accuracy: 0.9071
Epoch 8/30
1002/1002 [=====] - 78s 77ms/step - loss: 0.1486 - accuracy: 0.9441 - val_loss: 0.1674 - val_accuracy: 0.9381
Epoch 9/30
1002/1002 [=====] - 77s 77ms/step - loss: 0.1432 - accuracy: 0.9509 - val_loss: 0.2478 - val_accuracy: 0.9071
Epoch 10/30
1002/1002 [=====] - 77s 77ms/step - loss: 0.1431 - accuracy: 0.9481 - val_loss: 0.1576 - val_accuracy: 0.9441
Epoch 11/30
1002/1002 [=====] - 77s 76ms/step - loss: 0.1279 - accuracy: 0.9558 - val_loss: 0.1391 - val_accuracy: 0.9466
Epoch 12/30
1002/1002 [=====] - 77s 77ms/step - loss: 0.1182 - accuracy: 0.9558 - val_loss: 0.1326 - val_accuracy: 0.9550
Epoch 13/30
1002/1002 [=====] - 77s 77ms/step - loss: 0.1111 - accuracy: 0.9570 - val_loss: 0.1350 - val_accuracy: 0.9456
Epoch 14/30
1002/1002 [=====] - 77s 77ms/step - loss: 0.1070 - accuracy: 0.9625 - val_loss: 0.1233 - val_accuracy: 0.9555
Epoch 15/30
1002/1002 [=====] - 77s 77ms/step - loss: 0.1029 - accuracy: 0.9634 - val_loss: 0.1191 - val_accuracy: 0.9535
Epoch 16/30
1002/1002 [=====] - 77s 77ms/step - loss: 0.1134 - accuracy: 0.9592 - val_loss: 0.1096 - val_accuracy: 0.9600
Epoch 17/30
1002/1002 [=====] - 77s 77ms/step - loss: 0.1005 - accuracy: 0.9650 - val_loss: 0.1143 - val_accuracy: 0.9565
Epoch 18/30
1002/1002 [=====] - 78s 77ms/step - loss: 0.1018 - accuracy: 0.9624 - val_loss: 0.1202 - val_accuracy: 0.9555
Epoch 19/30
1002/1002 [=====] - 78s 78ms/step - loss: 0.0956 - accuracy: 0.9686 - val_loss: 0.1150 - val_accuracy: 0.9580
Epoch 20/30
1002/1002 [=====] - 78s 78ms/step - loss: 0.0953 - accuracy: 0.9657 - val_loss: 0.1790 - val_accuracy: 0.9291
Epoch 21/30
1002/1002 [=====] - 76s 76ms/step - loss: 0.0978 - accuracy: 0.9658 - val_loss: 0.1773 - val_accuracy: 0.9341
Epoch 22/30
1002/1002 [=====] - 76s 76ms/step - loss: 0.0911 - accuracy: 0.9672 - val_loss: 0.1321 - val_accuracy: 0.9515
Epoch 23/30
1002/1002 [=====] - 78s 77ms/step - loss: 0.0812 - accuracy: 0.9709 - val_loss: 0.1352 - val_accuracy: 0.9520
Epoch 24/30
1002/1002 [=====] - 77s 77ms/step - loss: 0.0898 - accuracy: 0.9712 - val_loss: 0.1049 - val_accuracy: 0.9595
Epoch 25/30
1002/1002 [=====] - 77s 77ms/step - loss: 0.0807 - accuracy: 0.9732 - val_loss: 0.1191 - val_accuracy: 0.9560
Epoch 26/30
1002/1002 [=====] - 78s 78ms/step - loss: 0.0819 - accuracy: 0.9704 - val_loss: 0.0906 - val_accuracy: 0.9660
Epoch 27/30
1002/1002 [=====] - 78s 78ms/step - loss: 0.0818 - accuracy: 0.9701 - val_loss: 0.1057 - val_accuracy: 0.9635
Epoch 28/30
1002/1002 [=====] - 78s 78ms/step - loss: 0.0769 - accuracy: 0.9702 - val_loss: 0.0888 - val_accuracy: 0.9670
Epoch 29/30
1002/1002 [=====] - 79s 78ms/step - loss: 0.0792 - accuracy: 0.9692 - val_loss: 0.0959 - val_accuracy: 0.9640
Epoch 30/30
1002/1002 [=====] - 78s 78ms/step - loss: 0.0771 - accuracy: 0.9738 - val_loss: 0.0895 - val_accuracy: 0.9685
```

Eğitimden sonra modelimizde %96.85'lik bir verim yakaladık. Fotoğraf sayımız çok olduğundan gayet güzel bir sonuç yakaladık. Şimdi bu modeli kendi fotoğraflarımıza uygulayacağız.



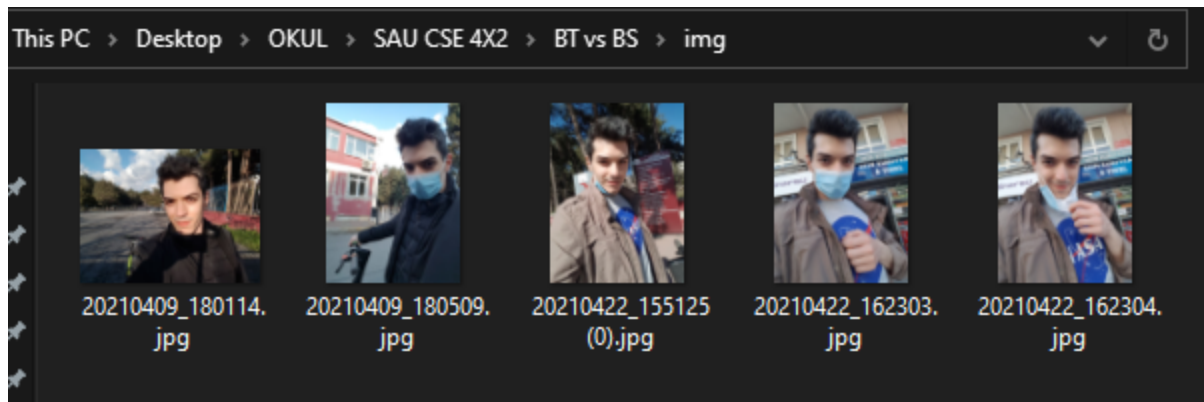
Görüntü İşleme

Modelimizi oluşturduktan sonra artık kendi fotoğraflarımı deneme vakti geldi.

```
from IPython.display import Image, display
for i in range (1,7):
    img_directory = str(i) + '.jpg'
    img_pred = image.load_img(img_directory, target_size = (70, 70))
    img_pred = image.img_to_array(img_pred)
    img_pred = np.expand_dims(img_pred, axis = 0)

    prediction = model.predict(img_pred)
    display(Image(img_directory,width= 150, height=150))
    print("\n")
    if(int(prediction[0][0]) == 0):
        print("Maske Takılmış.")
    else:
        print("Maske Takılmamış!")
    print("-----\n")
```

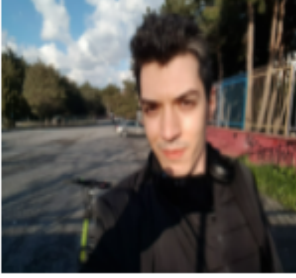
Kendi fotoğraflarımı ekleyip tahmin ettirerek maske takılıp takılmadığını göreceğim.



Birkaç maskeli ve maskesiz fotoğrafımı ekleyerek test edeceğim.



Fotoğraflarımı ekledikten sonra aşağıdaki gibi bir sonuç ile karşılaştım:



Maske Takılmamış!



Maske Takılmış.



Maske Takılmamış!



Maske Takılmamış!



Maske Takılmamış!

5 fotoğraftan 4'ünü doğru tahmin etti. Gayet güzel bir sonuç yakaladık.

Tensorflow ile görüntü işleme amacı ile maskeli ve maskesiz fotoğrafları modelleyerek maske takılı mı diye kontrol eden bir uygulama oluşturduk. Farklı fotoğraflar denedik, hatasız bir sonuç olmasa da gayet tatmin edici bir sonuç yakaladık.

Github: <https://github.com/onurgule/MaskeliMaskesiz>

