Yapay Sinir Ağı ile Çiçek Türü Sınıflandırması

Yapay Sinir Ağları (YSA), makine öğrenmesi alanında yaygın olarak kullanılan güçlü bir algoritmadır. Bu makalede, YSA kullanarak bir çiçek türü sınıflandırma problemi çözeceğiz. İlk adımda, veri kümesini inceleyeceğiz ve ardından veri önişleme adımlarını uygulayacağız. Daha sonra veriyi eğitim ve test kümelerine böleceğiz ve son olarak YSA modelini oluşturup eğiteceğiz.

1. Veri Kümesinin Adı ve Özellikleri

Veri kümesinin adı ve özellikleri: Veri kümesi 'iris.csv' olarak adlandırılır ve 150 satır ve 5 sütun içerir. Sütunlar sepal uzunluğu, sepal genişliği, petal uzunluğu, petal genişliği ve türdür. Türler setosa, versicolor ve virginica'dır. Veri kümesi, özelliklerine göre iris çiçeklerini sınıflandırmak için kullanılır.

2. Veri Önişleme

Veri önişleme adımları, veri kümesini modelimize uygun hale getirmek için kullanılır. İlk olarak, gerekli kütüphaneleri ve modülleri içe aktarıyoruz: pandas, numpy, matplotlib, seaborn, sklearn.preprocessing, sklearn.model_selection, keras.models ve keras.layers.

Veri kümesini "iris.csv" dosyasından okuyoruz ve bir DataFrame olan "df" değişkenine atıyoruz. Daha sonra, veri kümesinin şeklini ve bilgilerini yazdırarak veri hakkında önemli detaylara göz atıyoruz. Ayrıca, veri kümesinin betimsel istatistiklerini de görüntülüyoruz.

3. Veri Kümesinin Eğitim ve Test Olarak Bölünmesi

Veri kümesini eğitim ve test kümelerine bölmek için train_test_split fonksiyonunu kullanıyoruz. Veriyi %70 eğitim ve %30 test olacak şekilde ayırıyoruz. Bölme işleminden sonra, giriş ve çıkış değişkenlerini ayrı değişkenlere atıyoruz.

4. Yapay Sinir Ağı Yapısı

Yapay sinir ağının yapısı: Yapay sinir ağı, keras modülünden sequential sınıfı kullanılarak oluşturulur. İki yoğun katmanı vardır: ilki 8 birim ve ReLU aktivasyon fonksiyonuna sahip olan, ikincisi ise 3 birim ve softmax aktivasyon fonksiyonuna sahip olan. İlk katmanın giriş boyutu giriş özelliklerinin sayısına (4) dayanır ve ikinci katmanın çıkış boyutu çıkış sınıflarının sayısına (3) dayanır.

5. Yapay Sinir Ağında Kullanılan Fonksiyonlar

a. Toplama Fonksiyonu:

Yapay sinir ağında, yoğun katmanlarda kullanılan toplama fonksiyonu, ağırlıklar ve girişler arasındaki çarpımların toplamını hesaplar.

$$z_j = \sum_{i=1}^n w_{ij} x_i + b_j$$

Burada zj birim j için toplama fonksiyonunun çıktısıdır, wij önceki katmandaki birim i ile mevcut katmandaki birim j arasındaki ağırlıktır, xi önceki katmandaki birim i den giriştir, bj mevcut katmandaki birim j için yanlılıktır ve n önceki katmandaki birimlerin sayısıdır.

b. Gizli Katman Aktivasyon Fonksiyonu:

Bu fonksiyon, bir katmandaki her birimin toplama fonksiyonunun çıktısına doğrusal olmayan bir dönüşüm uygular. Bir birimin ne kadar aktive edildiğini veya ateşlendiğini belirler. Bu koddaki gizli katman için ReLU (düzeltilmiş doğrusal ünite) aktivasyon fonksiyonu kullanılır. Şöyle verilir:

$$a_j = \max(0, z_j)$$

Burada aj birim j için aktivasyon fonksiyonunun çıktısıdır ve zj birim j için toplama fonksiyonunun çıktısıdır.

c. Çıkış Katmanı Aktivasyon Fonksiyonu:

Çıkış katmanı için kullanılan aktivasyon fonksiyonu: Bu fonksiyon, bir katmandaki her birimin toplama fonksiyonunun çıktısına doğrusal olmayan bir dönüşüm uygular. Bir birimin nihai tahmine ne kadar katkıda bulunduğunu belirler. Bu koddaki çıkış katmanı için softmax aktivasyon fonksiyonu kullanılır. Şöyle verilir:

$$a_j = \frac{e^{z_j}}{\sum_{k=1}^m e^{z_k}}$$

Burada aj birim j için aktivasyon fonksiyonunun çıktısıdır, zj birim j için toplama fonksiyonunun çıktısıdır ve m çıkış katmanındaki birimlerin sayısıdır.

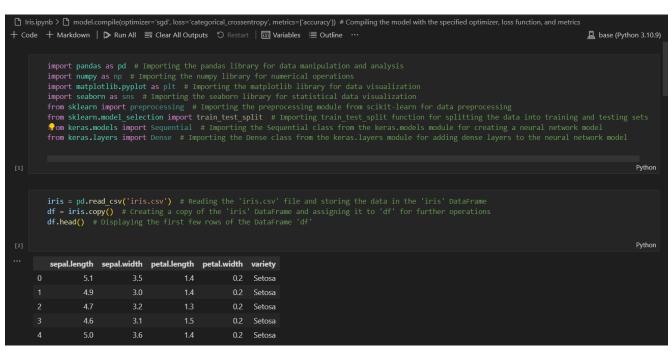
6. Öğrenme Algoritması

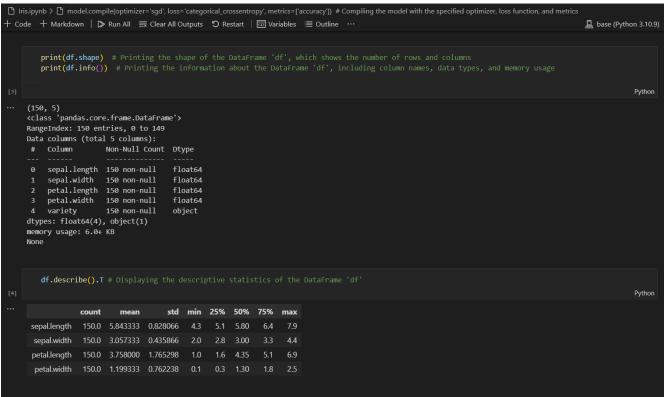
Öğrenme algoritması (pseudo code): Öğrenme algoritması, ağın eğitim verilerinde ne kadar iyi performans gösterdiğine bağlı olarak ağın ağırlıklarını ve yanlılıklarını günceller. Bu koddaki stokastik gradyan inişi (SGD) optimizasyonu öğrenme algoritması olarak kullanılır. Şöyle verilir:

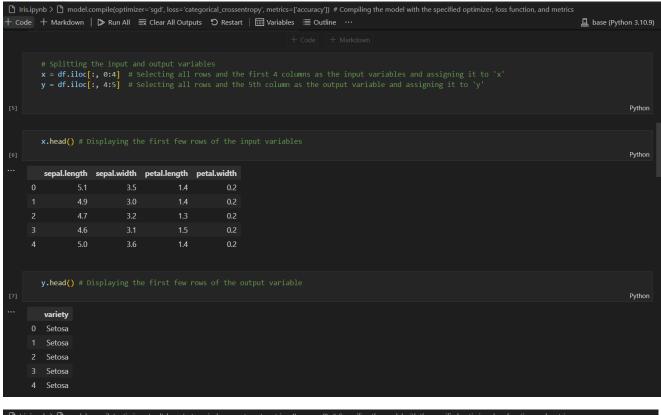
- o Ağırlıkları ve yanlılıkları rastgele başlat
- Her bir dönem için:
 - Eğitim verilerini karıştır
 - Her bir küme için:
 - Girişleri ağdan ileri yayınla ve çıktıları hesapla
 - Çıktılara ve gerçek etiketlere dayalı kayıp fonksiyonunu (kategorik çapraz entropi) hesapla
 - Hataları ağdan geri yayınla ve gradyanları hesapla
 - Ağırlıkları ve yanlılıkları gradyanlar ve öğrenme oranı kullanarak güncelle
- Son ağırlıkları ve yanlılıkları döndür

7. Seçilen Veri Kümesi için Yapay Sinir Ağı Kodu

Yukarıdaki adımları takip ederek oluşturduğumuz yapay sinir ağı modelinin Python kodu aşağıdaki gibidir:







```
| Integraph > | model.compile(optimizer-sydf, loss-categorical_crossentropy, metrics-[accuracy]) # Compiling the model with the specified optimizer, loss function, and metrics + Code + Markdown | D Run All | | Clear All Outputs | D Restat | | D Variables | D Outfine | P Output | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | D Run All | | D Run All | | D Run All | | D Run All | | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All | D Run All
```

```
🛅 trisipynb > 🛅 model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy']) # Compiling the model with the specified optimizer, loss function, and metrics
+ Code + Markdown | ▶ Run All 🗮 Clear All Outputs 🖰 Restart | 🛅 Variables 🗏 Outline …
                                                                                                                                                       base (Python 3.10.9)
         model = Sequential() # Creating a Sequential model
         model.add(Dense(8, input_dim=enrty_number, activation='relu')) # Adding a dense layer with 8 units, input dimension based on 'enrty_number', and Re
model.add(Dense(class_number, activation='softmax')) # Adding a dense layer with 'class_number' units and softmax activation function
         model.summary() # Displaying the summary of the model
     Model: "sequential"
      Layer (type)
                                     Output Shape
                                                                 Param #
      dense (Dense)
                                     (None, 8)
                                                                 40
       dense_1 (Dense)
                                     (None, 3)
     Total params: 67
     Trainable params: 67
     Non-trainable params: 0
                                                                                                                                                D ~
         model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy']) # Compiling the model with the specified optimizer, loss fund
         model.fit(x_train, y_train, batch_size=16, epochs=70) # Fitting the model to the training data with the specified batch size and number of epochs
                                                                                                                                                                   Python
                                                                                                                                                       base (Python 3.10.9)
         + Markdown | ▶ Run All 🗮 Clear All Outputs 🖰 Restart | 🖾 Variables 🗏 Outline
         model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy']) # Compiling the model with the specified optimizer, loss fund
     Epoch 1/70
                                           =] - 0s 3ms/step - loss: 1.4285 - accuracy: 0.0571
     7/7 [=
                                                0s 3ms/step - loss: 1.3340 - accuracy: 0.0762
     Epoch 3/70
                                                0s 4ms/step - loss: 1.2516 - accuracy: 0.0952
     7/7 [=
     Epoch 4/70
                                                0s 4ms/step - loss: 1.1774 - accuracy: 0.2190

    Os 4ms/step - loss: 1.1134 - accuracy: 0.2762

     7/7 [==
     Epoch 6/70
                                                0s 4ms/step - loss: 1.0530 - accuracy: 0.3714
     Epoch 7/70
                                             - 0s 4ms/step - loss: 0.9931 - accuracy: 0.4762
     7/7 [=
     Epoch 8/70
                                                0s 4ms/step - loss: 0.9394 - accuracy: 0.5810
     Epoch 9/70
     7/7 [=
                                          ==1 - 0s 3ms/step - loss: 0.8926 - accuracy: 0.6286
     Epoch 10/70
                                              - 0s 3ms/step - loss: 0.8520 - accuracy: 0.6667
     Epoch 11/70
     7/7 [=
                                          ==] - 0s 4ms/step - loss: 0.8161 - accuracy: 0.6857
     Epoch 12/70
                                         ===] - 0s 4ms/step - loss: 0.7839 - accuracy: 0.7333
     7/7 [=
     Epoch 13/70
     Epoch 69/70
                                    ======] - 0s 4ms/step - loss: 0.3645 - accuracy: 0.8476
     7/7 [==
```

```
tris.ipynb > 🖰 model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy']) # Compiling the model with the specified optimizer, loss function, and metric
Code + Markdown | ▶ Run All ➡ Clear All Outputs 与 Restart | ➡ Variables ➡ Outline …
                                                                                                                                                                                   ■ base (Python 3.10.9)
    Epoch 7/70
                               Epoch 8/70
                                                      0s 4ms/step - loss: 0.9394 - accuracy: 0.5810
    7/7 [==
                                                      0s 3ms/step - loss: 0.8926 - accuracy: 0.6286
    Epoch 10/70
                                                      0s 3ms/step - loss: 0.8520 - accuracy: 0.6667
    Epoch 11/70
                                               ===] - 0s 4ms/step - loss: 0.8161 - accuracy: 0.6857
    Epoch 12/70
                                     =======] - 0s 4ms/step - loss: 0.7839 - accuracy: 0.7333
    7/7 [=
    Epoch 13/70
    Epoch 69/70
                                              ===] - 0s 4ms/step - loss: 0.3645 - accuracy: 0.8476
                                      =======] - 0s 3ms/step - loss: 0.3626 - accuracy: 0.8571
    Output is truncated. View as a <u>scrollable element</u> or open in a <u>text editor</u>. Adjust cell output <u>settings</u>...
    <keras.callbacks.History at 0x23e59aeed70>
        result = model.evaluate(x_test, y_test, verbose=0) # Evaluating the model on the testing data and storing the results in the 'result' variable print("Accuracy: %.2f%%" % (result[1] * 100)) # Printing the accuracy rate by accessing the second element of 'result' and formatting it as a percentage print("Loss: %.2f%%" % (result[0] * 100)) # Printing the loss rate by accessing the first element of 'result' and formatting it as a percentage
                                                                                                                                                                                                   Python
    Accuracy: 86.67%
    Loss: 31.13%
```

Bu kod, veri kümesini okur, veri önişleme adımlarını uygular, veriyi eğitim ve test kümelerine böler, yapay sinir ağı modelini oluşturur ve eğitir, son olarak modelin performansını değerlendirir ve sonuçları yazdırır.

Bu makalede, yapay sinir ağları kullanarak çiçek türü sınıflandırması gerçekleştirmeyi öğrendik. Yapay sinir ağları, geniş bir uygulama yelpazesine sahip olduğundan, farklı problemleri çözmek için kullanılabilirler.

Öğretmen adı: Muhammed Karaaltun

Tarih: 11.06.2023

Hazırlayan Hasan Bari Akcan

KTO Karatay Üniversitesi Bilgisayar Mühendisliği