YSA ile Bina Enerji Verimliliği Analizi Detaylı Açıklama:

Yapay Sinir Ağları ile Bina Enerji Verimliliği Tahmini

Bu proje, bir binanın enerji verimliliğini (Isıtma Yükü ve Soğutma Yükü) tahmin etmek için bir Yapay Sinir Ağı (YSA) modeli oluşturmayı amaçlamaktadır. Model, binanın çeşitli fiziksel özelliklerini girdi olarak alır ve iki farklı enerji verimliliği değerini çıktı olarak tahmin eder.

Proje Adımları

1. **Veri Setini İndirme ve İnceleme:** Gerekli kütüphaneler yüklenir, veri seti okunur ve temel analizler yapılır.
2. **Veri Ön İşleme:** Eksik veriler temizlenir ve veri seti modele uygun hale getirilir.
3. **Eğitim ve Test Setlerini Ayırma:** Modelin performansını ölçebilmek için veri seti ikiye ayrılır.
4. **Yapay Sinir Ağı Modelini Oluşturma:** Keras Functional API kullanılarak çoklu çıkışlı bir YSA modeli tasarlanır.
5. **Modeli Derleme ve Eğitme:** Modelin öğrenme süreci yapılandırılır ve eğitim verileriyle eğitilir.
6. **Tahmin ve Değerlendirme:** Eğitilmiş model, test verileri üzerinde test edilir ve performansı R-kare (R^2) skoru ile değerlendirilir.

---

1. Kütüphanelerin Yüklenmesi ve Veri Setini Okuma

Bu bölümde projemizde kullanacağımız temel kütüphaneleri yüklüyoruz ve `.xlsx` formatındaki veri setimizi bir Pandas DataFrame'ine aktarıyoruz.

```
# VERİ SETİNİ İNDİRME VE İNCELEME KISMI
import numpy as np
```

```
import pandas as pd

data = pd.read_excel("C:/Users/Mehmet Ersolak/Documents/GitHub/Data_Science_Project/YSA İle Bİna Enerji Verimliliği/bina.xlsx")
data.head()
```

Açıklama:

- `import numpy as np`: NumPy, bilimsel hesaplamalar ve özellikle çok boyutlu diziler (array'ler) üzerinde verimli işlemler yapmak için kullanılan temel bir Python kütüphanesidir.
- `import pandas as pd`: Pandas, veri analizi ve manipülasyonu için güçlü araçlar sunar. Verileri `DataFrame` adı verilen tablo benzeri yapılar içinde tutarak işlememizi kolaylaştırır.
- `pd.read_excel(...)`: Bu fonksiyon, belirtilen yoldaki Excel dosyasını okur ve bir Pandas DataFrame nesnesine dönüştürür.
- `data.head()`: DataFrame'in ilk 5 satırını görüntüleyerek veri setinin yapısı, sütun adları ve içerdiği değerler hakkında hızlı bir önizleme yapmamızı sağlar.

---

2. Veri Seti Hakkında Genel Bilgileri Görüntüleme

Veri setinin içeriğini daha iyi anlamak için çeşitli Pandas fonksiyonlarını kullanarak temel istatistiksel bilgileri ve yapısal özetleri inceliyoruz.

```
data.info()

data.describe().T

data.count()
```

Açıklama:

- `data.info()`: Bu metod, DataFrame hakkında özet bir bilgi sunar. Sütun adlarını, her sütundaki `non-null` (boş olmayan) veri sayısını ve her sütunun veri tipini (Dtype) gösterir. Bu komut, eksik verileri (NaN) tespit etmek için oldukça kullanışlıdır.

- `data.describe().T`: Bu komut, veri setindeki sayısal sütunlar için temel istatistiksel bilgileri (sayım, ortalama, standart sapma, minimum, maksimum ve çeyreklik değerler) hesaplar. `.T` (transpose) ifadesi, tablonun daha okunaklı olması için satır ve sütunların yerini değiştirir.
- `data.count()`: Her sütundaki `non-null` (boş olmayan) değerlerin sayısını verir. `info()` metoduna benzer şekilde eksik veri kontrolü için kullanılır.

---

3. Veri Ön İşleme: Eksik Değerlerin Yönetimi

Modelimizin sağlıklı çalışabilmesi için veri setindeki eksik (`NaN`) değerleri temizliyoruz.

```
data_set = data.dropna() # Dropna kullanarak Nan değerlerini sildim
data_set = data_set.fillna(data_set.mean()) # data_setteki değerlerin ortalamas
ı ile Nan değerlerini doldurduk
data_set.info()
```

Açıklama:

- `data_set = data.dropna()`: Bu satır, içinde en az bir tane eksik değer (`NaN`) bulunan tüm satırları veri setinden siler. Bu, en basit ve en yaygın eksik veri yönetimi yöntemlerinden biridir.
- `# data_set.fillna(data_set.mean())`: Bu satır yorum satırı olarak bırakılmıştır ancak önemli bir alternatifi gösterir. Eğer verileri silmek yerine doldurmak isteseydik, bu kod her sütundaki eksik değerleri o sütunun ortalaması ile doldururdu. Genellikle bu iki yöntemden biri duruma göre tercih edilir.
- `data_set.info()`: Temizleme işleminden sonra `info()` fonksiyonunu tekrar çağırarak tüm sütunlardaki `non-null` sayılarının toplam satır sayısına eşit olduğunu ve böylece eksik veri kalmadığını teyit ederiz.

---

4. Eğitim ve Test Veri Setlerini Ayırma

Modelimizi eğitmek ve daha sonra performansını daha önce görmediği veriler üzerinde objektif bir şekilde ölçmek için veri setimizi eğitim ve test setleri olarak ikiye ayırıyoruz.

```
from sklearn.model_selection import train_test_split
X = data_set.iloc[:,0:8]
y = data_set.iloc[:,8:10]

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_stat
e=0)
```

Açıklama:

- `from sklearn.model_selection import train_test_split`: Scikit-learn kütüphanesinden veri setini kolayca bölmemizi sağlayan fonksiyonu import ediyoruz.
- `X = data_set.iloc[:,0:8]`: Girdi (bağımsız) değişkenlerimizi, yani özellikleri (`features`) seçiyoruz. `.iloc` ile indekse göre seçim yapıyoruz: tüm satırları (`:`) ve 0'dan 8'e kadar olan sütunları (ilk 8 sütun) `X` olarak atıyoruz.
- `y = data_set.iloc[:,8:10]`: Çıktı (bağımlı) değişkenlerimizi, yani hedefleri (`targets`) seçiyoruz. Tüm satırları ve 8 ile 9. sütunları (Isıtma ve Soğutma Yükü) `y` olarak atıyoruz.
- `train_test_split(...)`: Bu fonksiyon, `X` ve `y` verilerini belirtilen oranlarda karıştırarak böler:
  - `test_size=0.2`: Veri setinin %20'si test, %80'i eğitim için ayrılır.
  - `random_state=0`: Bölme işleminin her seferinde aynı şekilde yapılmasını sağlar. Bu, kod her çalıştığında aynı eğitim ve test setlerinin oluşmasını garantileyerek sonuçların tekrarlanabilir olmasını sağlar.

---

5. Verileri Yapay Sinir Ağı İçin Array Haline Getirme

TensorFlow/Keras gibi derin öğrenme kütüphaneleri, genellikle verileri Pandas DataFrame yerine NumPy dizileri (array) formatında bekler. Bu adımda veri setlerimizi bu formata dönüştürüyoruz.

```
import numpy as np
X_train_scaled = np.array(X_train)
X_test_scaled = np.array(X_test)
```

```
y_train_scaled = np.array(y_train)
y_test_scaled = np.array(y_test)

y_train_scaled = (y_train_scaled[:,0], y_train_scaled[:,1])
```

Açıklama:

- `np.array(...)`: Pandas DataFrame veya Series nesnelerini NumPy dizilerine dönüştürür.
- `y_train_scaled = (y_train_scaled[:,0], y_train_scaled[:,1])`: Bu satır çok önemlidir. Modelimizin iki ayrı çıkışı olduğu için (`first_output` ve `second_output`), Keras'a eğitim sırasında her bir çıkışa hangi hedef verinin karşılık geldiğini belirtmemiz gerekir. Bu kod, `y_train` dizisini iki ayrı vektöre ayırır (biri ilk hedef sütunu, diğeri ikinci hedef sütunu için) ve bunları bir demet (tuple) içine yerleştirir. Modelin `fit` fonksiyonu, `y` parametresi olarak bu formatı bekler.

---

6. Yapay Sinir Ağı Modelini Oluşturma

Bu bölümde, Keras Functional API'yi kullanarak modelimizin mimarisini tasarlıyoruz. Bu mimari, tek bir girdiden başlayıp ortak katmanlardan geçtikten sonra iki ayrı çıkış dalına ayrılan bir yapıya sahiptir.

```
from tensorflow.keras.layers import Input, Dense, Dropout
from tensorflow.keras.models import Model

# Giriş katmanı
input_layer = Input(shape=(X_train_scaled.shape[1],), name = 'Input_Layer')

# Ortak gizli katmanlar
common_path = Dense(units=128 , activation = 'tanh', name = 'First_Dense')(i
nput_layer)
common_path = Dropout(0.3)(common_path)
common_path = Dense(units = 128, activation='tanh', name = 'Second_Dens
e')(common_path)
common_path = Dropout(0.3)(common_path)
```

```
# 1. Çıkış Katmanı
first_output = Dense(units=1, name='First_Output_Last_Layer')(common_path)

# 2. Çıkış için ek katmanlar ve çıkış katmanı
second_output_path = Dense(units=64, activation='tanh', name='Second_Out
put_First_Dense')(common_path)
second_output_path = Dropout(0.3)(second_output_path)
second_output = Dense(units=1, name='Second_Output_Last_Layer')(second_
output_path)

# Modelin tamamının tanımlanması
model = Model(inputs=input_layer, outputs=[first_output, second_output])

print(model.summary())
```

Açıklama:

- `Input`: Modelin giriş katmanını tanımlar. `shape=(X_train_scaled.shape[1],)` ifadesi, giriş verisinin özellik sayısı kadar nörona sahip olacağını belirtir.
- `Dense`: Tam bağlantılı (fully connected) katmanlardır. Her nöron, bir önceki katmandaki tüm nöronlara bağlıdır.
  - `units`: Katmandaki nöron sayısını belirtir.
  - `activation='tanh'`: Aktivasyon fonksiyonudur. Nöronun çıktısını `1` ile `1` aralığına sıkıştırarak non-lineerlik ekler.
- `Dropout(0.3)`: Aşırı öğrenmeyi (overfitting) önlemek için kullanılan bir düzenlileştirme tekniğidir. Eğitim sırasında rastgele olarak nöronların %30'unu "kapatarak" modelin belirli nöronlara aşırı bağımlı hale gelmesini engeller.
- **Mimari Yapısı:**
  1. Giriş verisi, 128 nörondan oluşan iki `Dense` katmanından geçer. Bu katmanlar, her iki çıkış için de ortak özellikleri öğrenir.
  2. Daha sonra model ikiye ayrılır:
     - **Birinci Çıkış:** Ortak yoldan hemen sonra tek bir `Dense` katmanı ile doğrudan birinci hedefi (örn. Isıtma Yükü) tahmin eder.

    - **İkinci Çıkış:** Ortak yoldan sonra 64 nöronlu ek bir gizli katmandan daha geçerek ikinci hedef (örn. Soğutma Yükü) için daha karmaşık ve özel bir öğrenme süreci yürütür.
- `Model(inputs=..., outputs=...)`: Modelin girişini ve iki ayrı çıkışını belirterek mimariyi tamamlar.
- `model.summary()`: Oluşturulan modelin mimarisini, katmanlarını, çıktı şekillerini ve eğitilebilir parametre sayısını özetleyen bir tablo yazdırır.

---

7. Modeli Derleme (Compile)

Modeli eğitmeye başlamadan önce, öğrenme sürecini yapılandırmamız gerekir. Bu adımda optimize edici, kayıp fonksiyonu ve performans metrikleri belirlenir.

```
import tensorflow as tf

optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001, clipnorm=1.0)

model.compile(optimizer=optimizer,
              loss={'First_Output_Last_Layer': 'mse',
                    'Second_Output_Last_Layer': 'mse'},
              metrics={'First_Output_Last_Layer': tf.keras.metrics.RootMeanSquare
dError(),
                       'Second_Output_Last_Layer': tf.keras.metrics.RootMeanSquared
Error()})
```

Açıklama:

- `optimizer`: Modelin ağırlıklarını nasıl güncelleyeceğini belirleyen algoritmadır. `Adam` popüler ve genellikle iyi sonuç veren bir optimize edicidir.
    - `learning_rate=0.0001`: Öğrenme oranı, modelin ağırlıklarını her adımda ne kadar değiştireceğini kontrol eder. Düşük bir değer, daha yavaş ama potansiyel olarak daha kararlı bir öğrenme sağlar.
- `loss`: Kayıp fonksiyonu. Modelin tahminlerinin gerçek değerlerden ne kadar saptığını ölçer. Eğitimin amacı bu değeri minimize etmektir.

    - `'mse'` (Mean Squared Error - Ortalama Karesel Hata): Regresyon problemlerinde yaygın olarak kullanılan bir kayıp fonksiyonudur. Hataların karesini alarak büyük hataları daha fazla cezalandırır.
    - Her çıkış katmanı için ayrı bir kayıp fonksiyonu belirtmek amacıyla sözlük (dictionary) yapısı kullanılmıştır.
- `metrics`: Metrikler, eğitim sırasında modelin performansını insan gözüyle daha anlaşılır bir şekilde takip etmek için kullanılır. Kayıp fonksiyonundan farklı olarak, metrikler modelin ağırlıklarını güncellemek için kullanılmaz.
    - `RootMeanSquaredError` (Kök Ortalama Karesel Hata): MSE'nin karekökü dür. Hatanın, hedef değişkenle aynı birimde olmasını sağlayarak yorumlamayı kolaylaştırır.

---

8. Erken Durdurma (Early Stopping) ve Modeli Eğitme

Modelin gereğinden fazla eğitilerek ezber yapmasını (overfitting) önlemek ve en iyi performansı gösterdiği noktada eğitimi durdurmak için `EarlyStopping` mekanizmasını kullanıyoruz ve `fit` komutuyla modeli eğitiyoruz.

```
from tensorflow.keras.callbacks import EarlyStopping

earlyStopping_callback = EarlyStopping(monitor='val_loss',
                                       patience=10,
                                       verbose=1)

history = model.fit(x= X_train_scaled,
                    y = y_train_scaled,
                    epochs=500,
                    verbose=0,
                    batch_size=10,
                    validation_split=0.3,
                    callbacks=[earlyStopping_callback])
```

Açıklama:

- `EarlyStopping`: Bu bir "callback" (geri arama) fonksiyonudur ve eğitim sürecinin belirli noktalarında devreye girer.
  - `monitor='val_loss'`: İzlenecek değer olarak doğrulama (validation) setindeki kayıp değerini belirler.
  - `patience=10`: Doğrulama kaybında 10 epoch (dönem) boyunca bir iyileşme olmazsa eğitimi durdurur.
  - `verbose=1`: Eğitim durduğunda ekrana bir mesaj yazdırır.
- `model.fit()`: Modelin asıl eğitildiği fonksiyondur.
  - `x` ve `y`: Eğitim verileri ve hedefleri.
  - `epochs=500`: Modelin tüm eğitim verisi üzerinden maksimum 500 defa geçeceğini belirtir. Ancak `EarlyStopping` sayesinde bu sayıya ulaşmadan eğitim durabilir.
  - `batch_size=10`: Model, ağırlıklarını güncellemeden önce her adımda 10 veri örneğine bakacaktır.
  - `validation_split=0.3`: Eğitim verilerinin %30'unu her epoch sonunda modelin performansını kontrol etmek için doğrulama seti olarak ayırır. `EarlyStopping` bu seti izler.
  - `callbacks=[...]`: Eğitim sırasında kullanılacak `EarlyStopping` gibi geri arama fonksiyonlarını belirtir.

---

9. Tahmin Yapma ve Model Performansını Değerlendirme

Eğitim tamamlandıktan sonra, modelin daha önce hiç görmediği test verileri (`X_test_scaled`) üzerinde tahminler yaptırıyor ve bu tahminlerin ne kadar başarılı olduğunu R-kare (R^2) skoru ile ölçüyoruz.

```
y_predict = np.array(model.predict(X_test_scaled))

from sklearn.metrics import r2_score
print("İlk Çıkışın r2 Socre: ", r2_score(y_test_scaled[:,0], y_predict[0,:].flatten()))
```

```
print("İkinci Çıkışın r2 Socre: ", r2_score(y_test_scaled[:,1], y_predict[1,:].fla
tten()))
```

Açıklama:

- `model.predict(X_test_scaled)`: Eğitilmiş modeli kullanarak test seti için tahminler üretir. Modelin iki çıkışı olduğu için `y_predict` iki elemanlı bir liste (her eleman bir tahmin dizisi) olarak döner.
- `r2_score`: R-kare (Belirlilik Katsayısı), regresyon modelinin başarısını ölçen bir metriktir.
  - **Anlamı:** Bağımlı değişkendeki (hedef) varyansın ne kadarının bağımsız değişkenler (özellikler) tarafından açıklandığını gösterir.
  - **Değeri:** Genellikle 0 ile 1 arasında bir değer alır. 1'e ne kadar yakınsa, model o kadar başarılıdır. Örneğin, 0.90'lık bir R^2 skoru, hedef değişkendeki değişkenliğin %90'ının model tarafından açıklandığı anlamına gelir.
- `y_test_scaled[:,0]`: Test setindeki gerçek değerlerin birinci sütunu.
- `y_predict[0,:].flatten()`: Modelin tahminlerinin birinci çıkışa ait olanları. `.flatten()` metodu, diziyi tek boyutlu hale getirerek `r2_score` fonksiyonuyla uyumlu olmasını sağlar.
- İkinci `print` satırı da aynı işlemi ikinci çıkış ve ikinci hedef sütunu için tekrarlar.