KÜTAHYA SAĞLIK BİLİMLERİ ÜNİVERSİTESİ
MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ

YAPAY ZEKA DERSİ

Recurrent Neural Networks(RNN) Kullanarak Hava Durumu Tahmini

Yusuf KIZILGEDİK **2118121003**

Thursday 25th April, 2024

Özet

Bu çalışmada, hava durumu tahmininde kullanılan yapay zeka teknikleri ve özellikle tekrarlayan sinir ağlarının (RNN) rolü üzerinde durulmuştur. Çalışmanın amacı, geçmiş hava durumu verilerini kullanarak gelecekteki hava durumunu tahmin etmek için RNN tabanlı bir yapay zeka uygulaması geliştirmektir. Hava durumu tahmini için RNN'nin neden tercih edildiği, kullanılan algoritmalar ve uygulama performansı detaylı bir şekilde açıklanmıştır. Elde edilen sonuçlar, bu uygulamanın hava durumu tahmininde başarılı olduğunu göstermiştir. Ayrıca, diğer algoritmalarla yapılan karşılaştırmalar da değerlendirilmiştir.

Anahtar Kelimeler

Hava Durum Tahmin Uygulaması,Yapay Zeka,RNN,LSTM

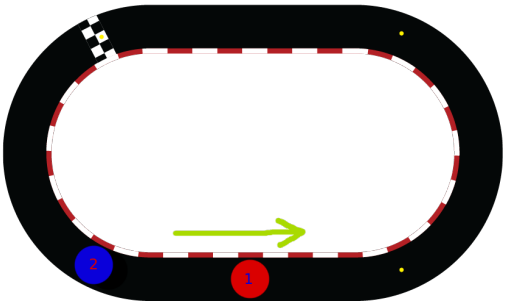
1. **Giriş**

   Bu çalışma, hava durumu tahmini için yapay zeka ve özellikle tekrarlayan sinir ağlarının kullanımını incelemektedir. Hava durumu tahmini, zamansal verilerin analizi gerektiren karmaşık bir alandır. Tekrarlayan sinir ağlarının, zamansal ilişkileri daha iyi anlamak için kullanılabilecek çeşitli türleri vardır. Hava durumu tahmini konusunda RNN'nin avantajları ve bu çalışmanın neyi başarmayı hedeflediği burada açıklanmaktadır.

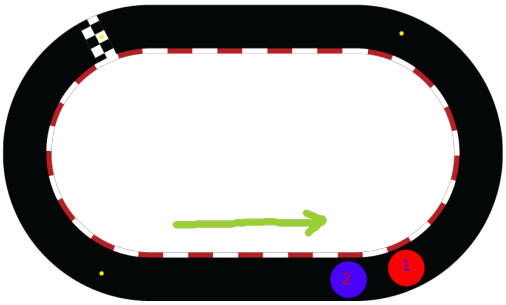
   Bu çalışmada, hava durumu tahmininde yapay zeka uygulamalarının önemine dair önceden yapılan çalışmalar ve bu çalışmanın neden önemli olduğu da vurgulanmaktadır. Önceki çalışmalardaki eksiklikler ve bu çalışmanın çözmeyi hedeflediği problemler üzerinde durulmaktadır. Bu çalışmanın amacı, hava durumu tahmininde derin öğrenme tekniklerini kullanarak daha doğru tahminler elde etmektir. [2]

2. **Neden RNN?**

   Hava durumu tahmini gibi zamansal verilerin analizi için RNN (Recurrent Neural Networks), standart sinir ağlarına kıyasla daha uygun bir seçenek olabilir. Standart sinir ağları, veriler arasındaki ilişkileri analiz etmek için o anki verilere dayanır ve geçmiş verilerin zaman bağlamını göz ardı eder. Örneğin, bir yarış pistindeki iki aracın yarışını ele alalım: birinci görselde kırmızı araç, mavi araca kıyasla oldukça önde ve birincil konumda ilerlemektedir. Ancak, ikinci görselde kırmızı araç hala birinci sıradadır, ancak mavi araçla arasındaki mesafe oldukça azalmıştır. Bu noktada, standart sinir ağları, her iki görseli ayrı ayrı değerlendirir ve kırmızı aracın kazanacağını tahmin eder. Ancak, RNN, geçmiş verilerin zaman bağlamını dikkate alarak analiz yapar. Dolayısıyla, ikinci görseldeki durumda, mavi aracın mesafeyi kapatma eğiliminde olduğunu ve sonucun değişebileceğini tahmin eder. Bu nedenle, zaman serileri gibi zamansal verilerle çalışırken RNN, zaman bağlamını dikkate alarak daha doğru tahminlerde bulunabilir.



(a) 1.Göresel



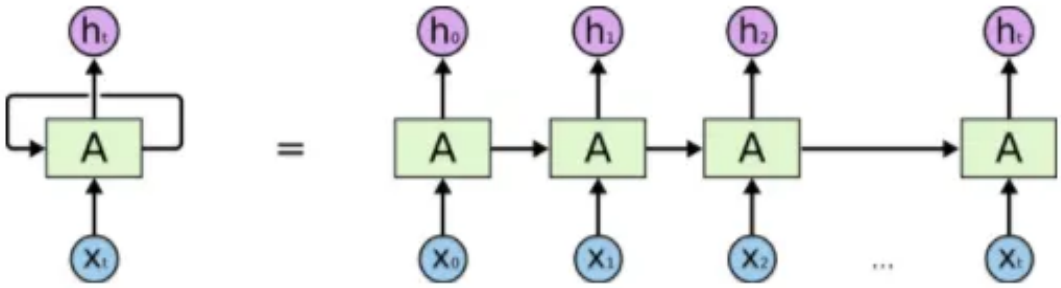
(b) 2.Görsel

3. **RNN Türleri**

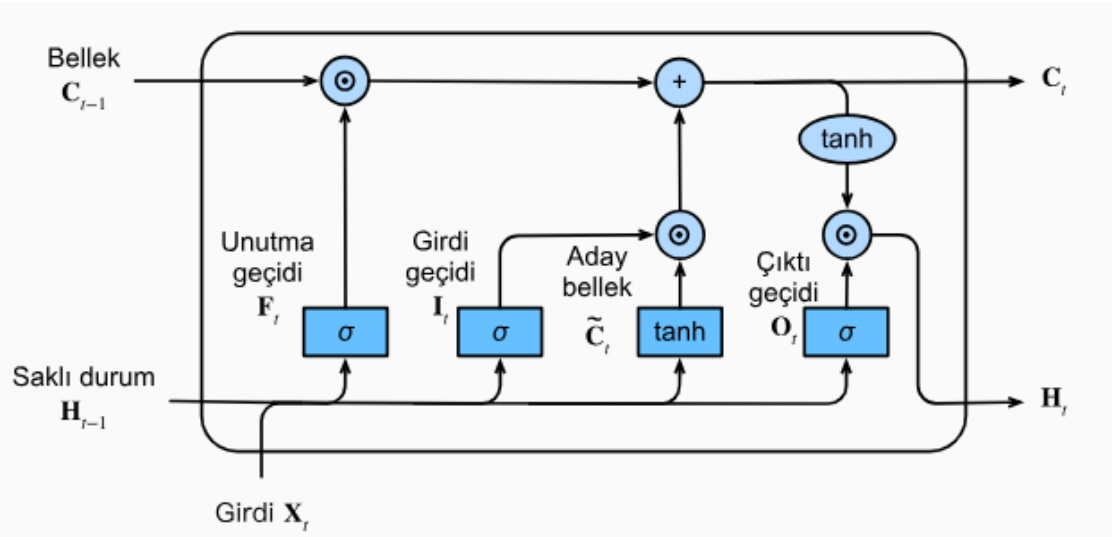
- **Vanilla RNN (Temel RNN):** Basitçe geçmiş zaman adımlarını dikkate alarak bir sonraki adımın tahminini yapar. Ancak, zamanla kaybolan (vanishing gradients) sorunu nedeniyle uzun vadeli bağlantıları yakalamakta zorlanabilir.
- **Long Short-Term Memory (LSTM):** LSTM, zaman serisi verilerinde uzun vadeli bağımlılıkları daha etkili bir şekilde modellemek için tasarlanmıştır. Hava durumu tahmini gibi zaman serisi problemleri için çok kullanışlıdır.
- **Gated Recurrent Unit (GRU):** GRU, LSTM'e benzer şekilde uzun vadeli bağımlılıkları ele almak için tasarlanmıştır. Ancak, LSTM'e kıyasla daha az parametreye sahiptir, bu da eğitim sürecini hızlandırabilir.
- **Bidirectional RNN (BRNN):** BRNN, geçmiş ve gelecek zaman adımlarını ayrı ağlarda işleyerek, her bir zaman adımında hem önceki hem de sonraki verileri dikkate alır. Bu şekilde, mevcut zamandaki tahminler için daha kapsamlı bir bağlam sağlar.
- **Deep RNN:** Birden fazla katmana (layer) sahip RNN'lerdir. Daha derin yapılar, daha karmaşık zaman serisi ilişkilerini yakalamak için kullanılabilir.
- **CNN-RNN Hibrit Modeller:** Bazı durumlarda, zaman serisi verilerini işlemek için Convolutional Neural Networks (CNN) ile RNN'leri birleştiren modeller de etkilidir. Özellikle uzaysal yapıyı (örneğin, hava durumu haritaları) dikkate almak istediğinizde, bu tür hibrit modeller kullanışlı olabilir.

4. **Kullandığımız RNN(LSTM)**
   Long Short-Term Memory (LSTM), hava durumu tahmini gibi zaman serisi problemlerinde oldukça etkili olan bir tür Recurrent Neural Network (RNN)'dir. LSTM, zaman serisi verilerinde uzun vadeli bağımlılıkları daha etkili bir şekilde modellemek için tasarlanmıştır.
   Hava Durumu Tahmin projemde tercih etmemim sebeleri şunlardır:

   - Uzun Vadeli Bağımlılıkların Modellemesi: Hava durumu, geçmiş hava koşullarının mevcut ve gelecek durumu etkilediği bir süreçtir. LSTM'nin hafıza hücreleri, bu uzun vadeli bağımlılıkları daha etkili bir şekilde modellemek için idealdir.
   - Zaman Serisi Verilerinin Karmaşıklığı: Hava durumu verileri genellikle karmaşıktır ve birçok farklı değişkeni içerir (sıcaklık, nem, rüzgar hızı vb.). LSTM, bu karmaşıklığı ele almak için esnek bir yapı sunar.
   - Esnek Veri Girişi İşleme: LSTM, farklı zaman adımlarındaki veri girişlerini ve çıkışlarını işlemek için esnek bir yapıya sahiptir. Bu, hava durumu tahmini gibi zaman serisi problemlerinde çok önemlidir çünkü veriler genellikle düzensiz ve değişken bir şekilde gelir.
   - Gradient Kaybının Azaltılması: LSTM'nin vanishing gradient sorununu azaltma yeteneği, uzun vadeli tahminlerde daha güvenilir sonuçlar elde etmeyi sağlar.



Şekil 2: RNN [2]



Şekil 3: LSTM [3]

```python
# Bu fonksiyon bir LSTM hücresini tanimlar.
def LSTMCELL(prev_ct, prev_ht, Xt):



    combine = prev_ht + Xt #1-İlk olarak, önceki gizli durum ile mevcut giriş birleştirilir.

    # Unutma kapisini kontrol eder.
    ft = forget_layer(combine)

    # Aday hücre durumunu hesaplar.
    candidate = candidate_layer(combine) #Aday, hücre durumuna eklemek için olasi değerleri tutar.


    # Giriş kapisini kontrol eder.
    it = input_layer(combine)

    # Mevcut zaman dilimindeki bellek durumunu hesaplar.
    Ct = prev_ct * ft + candidate * it

    # Çikiş kapisini kontrol eder.
    Ot = output_layer(combine)

    # Mevcut zaman dilimindeki gizli durumunu hesaplar.
    ht = Ot * tanh(Ct)

    # `Ct` ve `ht`'yi fonksiyonun çiktisi olarak döndürür.
    return ht, Ct

# Başlangiç bellek ve gizli durumlari.
ct = [0, 0, 0]
ht = [0, 0, 0]

# Her zaman diliminde fonksiyonu çağirir.
for Xt in inputs:
    Ct, ht = LSTMCELL(ct, ht, Xt)
```

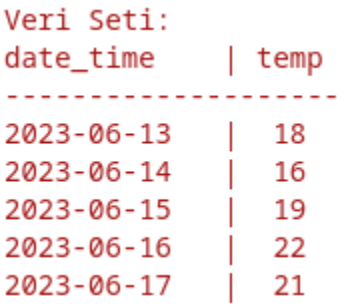
Şekil 4: LSTM python pseudo code [4]

5. LSTM ile Hava Durumu Tahmini Kod

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Excel dosyasını okuyun
df = pd.read_excel('sakarya2018-2024.xlsx')

# Veri setini işleyin
df['date_time'] = pd.to_datetime(df['date_time'])
df.set_index('date_time', inplace=True)

# Veriyi ölçeklendirin
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df)

# Eğitim ve test veri setlerini ayırın
X = scaled_data[:-1]
y = scaled_data[1:]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)

# Eğitim ve test veri setlerini yeniden şekillendirin
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

# LSTM modelini oluşturun
model = Sequential([
    LSTM(50, activation='relu', input_shape=(X_train.shape[1], X_train.shape[2])),
    Dense(1)
])

model.compile(optimizer='adam', loss='mse')

# Modeli eğitin
model.fit(X_train, y_train, epochs=100, batch_size=1, verbose=1)

# Önümüzdeki 4 günün tahminini yapın
future_days = 4
future_dates = pd.date_range(start=df.index[-1], periods=future_days + 1)[1:]  # Bugünden itibaren 4 gün ileri

# Son gözlemi alın ve ölçeklendirin
last_data = scaled_data[-1]

# Tahminleri depolamak için bir dizi oluşturun
predicted_values = []

# Gelecekteki her bir gün için tahmini yapın
for _ in range(future_days):
    # Ölçeklendirilmiş veriyi yeniden şekillendirin
    X_new = np.array(last_data.reshape(1, -1, 1))
    # Tahmini yapın
    prediction = model.predict(X_new)
    # Ölçeklendirmeyi tersine çevirin
    prediction = scaler.inverse_transform(prediction)
    # Tahmini değeri saklayın
    predicted_values.append(prediction[0][0])
    # Tahmin edilen değeri son veriye ekleyin ve ilk girdiyi kaldırın
    last_data = np.append(last_data[1:], prediction[0][0])

# Tahmin edilen sıcaklık değerlerini ekrana yazdırın
for date, temp in zip(future_dates, predicted_values):
    print(f"{date.date()}: {temp:.2f} °C")
```

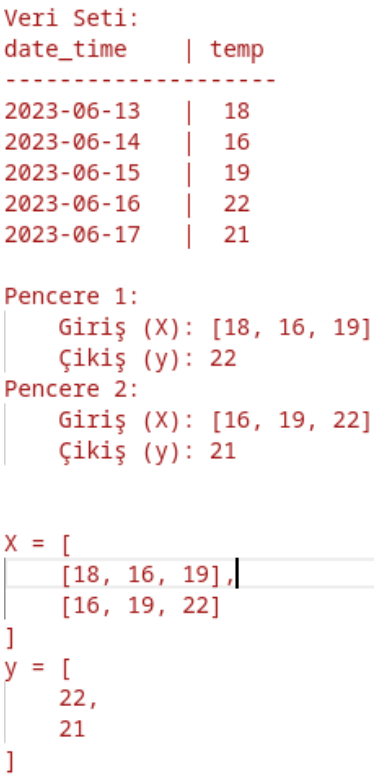
Şekil 5: Hava Durumu Tahmini ilk Kod [1]

- Yapay sinir ağlarının türlerinden biri olan Uzun Kısa Süreli Bellek (LSTM) modelini kullanan bir Python kodu yazdım. Başlangıçta, bu kod doğru tahminler yapmıyordu ve sonuçlar beklenenin oldukça dışında, tuhaf değerler üretiyordu. Bu durumu düzeltmek ve algoritmanın performansını artırmak için bir dizi iyileştirme yaptım.



Şekil 6: Veri Seti

- **Veri Ön İşleme** Herhangi bir makine öğrenimi projesinde olduğu gibi, veri ön işleme adımı hava durumu tahmini için de kritiktir. İlk adım olarak, hava durumu veri setimizi alırız. Örneğimizde, hava durumu ölçümleri Sakarya şehrimizin 2018-2024 yılları arasındaki tarih bilgilerini içeren bir veri seti kullanıyoruz. Veri setimizi Pandas kütüphanesi aracılığıyla yükleriz ve tarih saat sütununu indeks olarak ayarlarız. Daha sonra, veriyi ölçeklendirme işlemine tabi tutarız. Bu, tüm değerleri belirli bir aralığa (genellikle [0, 1] aralığına) yeniden ölçeklendirir ve modelin daha iyi performans göstermesine yardımcı olur.[5]



Şekil 7: Pencere Veri

- **Veriyi Window Haline Getirme** LSTM gibi zaman serisi modelleriyle çalışırken, veriyi pencere (window) haline getirmek yaygın bir uygulamadır. Bu, belirli bir zaman dilimindeki verileri bir araya getirerek, modelin zamansal ilişkileri daha iyi öğrenmesine olanak tanır. Örneğin, son üç günün hava durumu verilerini kullanarak, bir sonraki günün sıcaklık tahminini yapabiliriz. Bu işlem, bir veri penceresi boyunca belirli bir adımda gezinerek gerçekleştirilir. Yukarıdaki görselde, verinin pencere haline getirilmesini gösteren bir örnek bulunmaktadır. Her adımda, belirli bir pencere boyutu alınır ve bu pencere boyunca sıcaklık verileri toplanır. Bu, X ve y verilerini oluşturur ve modelin eğitimi için kullanılır.
- **Dropout** Aşırı öğrenmeyi önlemek için yaygın olarak kullanılan bir tekniktir. Bu teknik, rastgele belirlenen bir oranda ağdaki nöronların çıkartılmasını sağlar. Bu, ağın genelleştirme yeteneğini artırır ve aşırı uyumun önüne geçer. Modelimizde Dropout katmanları kullanarak, her eğitim döneminde belirli bir oranda nöronları rastgele devre dışı bırakırız.

- **EarlyStopping** Modelin eğitimini belirli bir kriter karşılandığında durduran bir geri çağrıdır. Bu, modelin aşırı uyuma meyilli olduğu durumlarda ağın eğitimini zamanında durdurarak, ağın genelleme yeteneğini artırır. Örneğin, modelin kaybı artık azalmıyorsa veya doğrulama kaybı artmaya başlarsa, eğitimi durdurabiliriz.
- **Model Karmaşıklığının Artırılmas** Modelin karmaşıklığını artırmak için daha fazla LSTM katmanı eklendi. LSTM katmanları, zaman serisi verilerindeki karmaşık ilişkileri yakalamak için kullanılır. Modelin karmaşıklığını artırmak, daha fazla öğrenme kapasitesi sağlayabilir ve daha karmaşık veri yapılarını daha iyi modelleyebilir.
- **Daha Fazla Epoch Sayısının Kullanılması** Modelin eğitim sürecinde daha fazla epoch (iterasyon) sayısı kullanıldı. Daha fazla epoch, modelin daha fazla öğrenme fırsatı elde etmesini sağlar. Bu, modelin daha iyi uyum sağlamasını ve daha doğru tahminler üretmesini sağlayabilir.

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping

# Excel dosyasını okuyun
df = pd.read_excel('sakarya2018-2024.xlsx')

# Veri setini işleyin
df['date_time'] = pd.to_datetime(df['date_time'])
df.set_index('date_time', inplace=True)

# Veriyi ölçeklendirin
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df)

# Öğrenme ve test veri setlerini ayırın
train_size = int(len(scaled_data) * 0.8)
train_data, test_data = scaled_data[:train_size], scaled_data[train_size:]

# Veriyi pencerele (window) haline getirin
def create_dataset(data, window_size):
    X, y = [], []
    for i in range(len(data) - window_size):
        X.append(data[i:i+window_size])
        y.append(data[i+window_size])
    return np.array(X), np.array(y)

window_size = 3  # Pencere boyutu
X_train, y_train = create_dataset(train_data, window_size)
X_test, y_test = create_dataset(test_data, window_size)

# LSTM modelini oluşturun
model = Sequential([
    LSTM(100, activation='relu', return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])),
    Dropout(0.2),
    LSTM(100, activation='relu', return_sequences=True),
    Dropout(0.2),
    LSTM(100, activation='relu', return_sequences=True),
    Dropout(0.2),
    LSTM(100, activation='relu'),
    Dropout(0.2),
    Dense(1)
])

model.compile(optimizer='adam', loss='mse')
# Erken durdurma callback'i ekleyerek modelin eğitimini iyileştirin
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Modeli eğitin
history = model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=1, validation_data=(X_test, y_test), callbacks=[early_stopping])

# Önümüzdeki 7 günün tahminini yapın
future_days = 7
future_dates = pd.date_range(start=df.index[-1], periods=future_days + 1)[1:]  # Bugünden itibaren 4 gün ileri

# Son pencereyi alın
last_window = scaled_data[-window_size:].reshape(1, window_size, 1)

# Tahminleri yapın
predicted_values = []
for _ in range(future_days):
    prediction = model.predict(last_window)[0]
    predicted_values.append(prediction)
    last_window = np.append(last_window[:, 1:, :], prediction.reshape(1, 1, 1), axis=1)

# Tahmin edilen sıcaklık değerlerini ölçekten çıkarın
predicted_values = scaler.inverse_transform(predicted_values)

# Tahmin edilen sıcaklık değerlerini ekrana yazdırın
for date, temp in zip(future_dates, predicted_values):
    print(f"{date.date()}: {temp[0]:.2f} °C")
```

Şekil 8: Hava Durumu Tahmini geliştirilmiş Kod [1]

- Modelde gerçekleştirdiğim iyileştirmelerden sonra, sonuçların ne kadar geliştiğini belirlemek için eski ve yeni sonuçları karşılaştırdım.

date time	temp
2023-11-24	9
2023-11-25	9
2023-11-26	10
2023-11-27	2
2023-11-28	2

Table 1: Gerçek Veri

date time	temp
2023-11-24	8.97
2023-11-25	565.90
2023-11-26	27567.37
2023-11-27	1292014.62
2023-11-28	60554036.00

(a) 1.Kod Sonuçları

date time	temp
2023-11-24	8.64
2023-11-25	9.01
2023-11-26	9.44
2023-11-27	9.65
2023-11-28	9.90

(b) 2.Kod Sonuçları

Son geliştirmelerimiz sonucunda, modelimizdeki çıktılarda önemli bir doğruluk artışı elde ettik.

References

[1] Weather forecasting with recurrent neural networks. https://medium.com/analytics-vidhya/weather-forecasting-with-recurrent-neural-networks-1eaa057d70c3. Erişim Tarihi: 20 Mart 2024.

[2] Deep Learning Türkiye. Rnn nedir? nasıl Çalışır? https://medium.com/deep-learning-turkiye/rnn-nedir-nas%C4%B1l-%C3%A7al%C4%B1%C5%9F%C4%B1r-9e5d572689e1, 2018. Erişim Tarihi: 20 Mart 2024.

[3] Dive into Deep Learning. Lstm (uzun kısa süreli bellek). https://tr.d2l.ai/chapter_recurrent-modern/lstm.html. Erişim Tarihi: 20 Mart 2024.

[4] Towards Data Science. Illustrated guide to lstms and grus: A step-by-step explanation. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb Erişim Tarihi: 20 Mart 2024.

[5] UlaşAV. Hava durumu Ölçüm değerleri. https://ulasav.csb.gov.tr/dataset/42-hava-durumu-olcum-degerleri, 2023. Erişim Tarihi: 20 Mart 2024.