Aşırı Öğrenme Makinesi (ELM) Üzerine Bir İnceleme

Konu: Aşırı Öğrenme Makinesi (Extreme Learning Machine - ELM) Nedir? **Hazırlayan:** ilknur şevval yılmaz-02220224040 **Tarih:** 29.12.2024

1. Mimari ve Genel Yapı

Aşırı Öğrenme Makinesi (ELM), yapısal olarak bildiğimiz Tek Gizli Katmanlı İleri Beslemeli Sinir Ağlarına (SLFN) dayanır. Ancak işleyiş mantığı, özellikle Geri Yayılım (Backpropagation) gibi klasik yöntemlerle kıyaslandığında, öğrenme hızı ve parametre optimizasyonu açısından temelden farklıdır.

ELM mimarisi üç ana katmandan oluşur:

1. **Giriş Katmanı:** Verinin sisteme girdiği katmandır. n adet giriş nöronu bulunur ve veriyi herhangi bir işlem yapmadan gizli katmana iletir.
2. **Gizli Katman:** ELM'nin en karakteristik özelliğidir. Bu katmandaki $\tilde{N}$ adet nöronun giriş ağırlıkları (w) ve eşik değerleri (b), eğitim verisinden bağımsız olarak **rastgele** atanır. Eğitim süresince bu değerler sabit tutulur (güncellenmez).
3. **Çıktı Katmanı:** m adet nöron içerir. Gizli katman ile çıktı katmanı arasındaki ağırlıklar (β), iteratif bir süreç yerine analitik bir yöntemle hesaplanır.

2. Modeli ve Temel Denklemleri

ELM'nin teorik altyapısı, iteratif bir optimizasyon probleminden ziyade, lineer bir denklem sisteminin analitik çözümüne dayanmaktadır.

2.1. Matematiksel Gösterim

N adet birbirinden farklı örnekten oluşan bir veri seti $\{(x_i, t_i)\}_{i=1}^{N}$ ele alalım. Burada $x_i \in \mathbb{R}^n$ giriş vektörünü, $t_i \in \mathbb{R}^m$ ise hedef çıktı vektörünü temsil eder.

$\tilde{N}$ adet gizli nörona ve $g(x)$ aktivasyon fonksiyonuna sahip bir SLFN'in matematiksel modeli şu denklem ile ifade edilir:

$$\sum_{i=1}^{\tilde{N}} \beta_i g(w_i \cdot x_j + b_i) = o_j, \quad j = 1, \ldots, N$$

Bu denklemdeki değişkenlerin tanımları şöyledir:

- $w_i = [w_{i1}, \ldots, w_{in}]^T$: i. gizli nöronu giriş nöronlarına bağlayan **ağırlık vektörü**.
- $\beta_i = [\beta_{i1}, \ldots, \beta_{im}]^T$: i. gizli nöronu çıktı nöronlarına bağlayan **ağırlık vektörü**.
- b_i: i. gizli nöronun **eşik (bias)** değeri.
- o_j: Ağın ürettiği çıktı.

2.2. Matris Formülasyonu ($\mathbf{H}\beta = \mathbf{T}$)

ELM algoritmasının amacı, eğitim hatasını sıfıra indirmek, yani $\sum_{j=1}^{N} \|o_j - t_j\| = 0$ olmalıdır. Bu durumda yukarıdaki N adet denklem, matris formunda şu şekilde yazılır:

$$\mathbf{H}\beta = \mathbf{T}$$

Bu denklemdeki matrislerin açık yapıları aşağıdaki gibidir:

1. Gizli Katman Çıktı Matrisi ($\mathbf{H}$): Huang ve ark. (2006) tarafından tanımlandığı üzere, bu matris gizli katmandaki nöronların girişlere verdiği tepkiyi (aktivasyonu) içerir. w ve b değerleri rastgele atanıp sabitlendiği için $\mathbf{H}$ **matrisi sabittir.**

$$\mathbf{H} = \begin{bmatrix} g(w_1 \cdot x_1 + b_1) & \cdots & g(w_{\tilde{N}} \cdot x_1 + b_{\tilde{N}}) \\ \vdots & \ddots & \vdots \\ g(w_1 \cdot x_N + b_1) & \cdots & g(w_{\tilde{N}} \cdot x_N + b_{\tilde{N}}) \end{bmatrix}_{N \times \tilde{N}}$$

2. Çıktı Ağırlıkları Matrisi (β): Hesaplanması gereken bilinmeyen parametre matrisidir.

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_{\tilde{N}}^T \end{bmatrix}_{\tilde{N} \times m}$$

3. Hedef Çıktı Matrisi ($\mathbf{T}$): Eğitim setindeki gerçek etiketleri içerir.

$$\mathbf{T} = \begin{bmatrix} t_1^T \\ \vdots \\ t_N^T \end{bmatrix}_{N \times m}$$

2.3. Analitik Çözüm ve Optimizasyon

Genellikle gizli nöron sayısı, eğitim örneği sayısından az seçildiği için ($\tilde{N} \ll N$), sistem aşırı belirlenmiş (overdetermined) bir sistemdir ve tam çözümü olmayabilir. Bu durumda ELM, hatayı minimize eden **En Küçük Kareler (Least Squares)** çözümünü arar:

$$\min_{\beta} \|\mathbf{H}\beta - \mathbf{T}\|$$

Bu problemin minimum normlu çözümü, $\mathbf{H}$ matrisinin **Moore-Penrose Genelleştirilmiş Tersi ($\mathbf{H}^\dagger$)** kullanılarak tek adımda elde edilir:

$$\hat{\beta} = \mathbf{H}^\dagger \mathbf{T}$$

Bu formül, ELM'nin eğitim sürecinin tamamını oluşturur. $\mathbf{H}^\dagger$ matrisinin hesaplanması, türev tabanlı yinelemeli yöntemlere kıyasla çok daha hızlıdır ve yerel minimuma takılma riskini ortadan

kaldırır.

3. Neden Önemli? (Kritik Özellikler)

ELM'yi literatürde öne çıkaran temel özellikler şunlardır:

- **Hız Faktörü:** Geri yayılım (Backpropagation) algoritmalarının binlerce iterasyonda (epoch) yaptığı işlemi, ELM tek bir matris işlemiyle gerçekleştirir.
- **Ayar Gerektirmemesi:** Öğrenme katsayısı (learning rate) veya momentum gibi hiperparametrelerin ayarlanmasına ihtiyaç duymaz.
- **Genelleme Yeteneği (Evrensel Yaklaşım):** Yeterli sayıda gizli nöron verildiğinde, ELM'nin her türlü sürekli fonksiyonu modelleyebildiği matematiksel olarak kanıtlanmıştır (Huang et al., 2006).

4. Uygulama: Fisher Iris Sınıflandırması

Teorik kısmı pekiştirmek için Fisher Iris veri seti üzerinde çalışan bir kod hazırladım. Hazır kütüphane yerine, yukarıdaki $\hat{\beta} = \mathbf{H}^{\dagger}\mathbf{T}$ denklemini doğrudan NumPy ile uyguladım.

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.metrics import accuracy_score, classification_report

class ELMClassifier:
    """
    ELM Algoritmasının İmplementasyonu.
    Temel Denklem: H * Beta = T
    Çözüm: Beta = pinv(H) * T
    """

    def __init__(self, hidden_neurons=20, random_state=42):
        self.hidden_neurons = hidden_neurons
        self.random_state = random_state
        self.input_weights = None
        self.biases = None
        self.beta = None

    def _sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def fit(self, X, y):
        # Tekrarlanabilirlik için seed
        np.random.seed(self.random_state)
        n_samples, n_features = X.shape

        # 1. ADIM: Giriş ağırlıkları (w) ve biasların (b) RASTGELE atanması
        # Literatüre göre bu değerler sabittir.
        self.input_weights = np.random.normal(size=(n_features, self.hidden_neurc
        self.biases = np.random.normal(size=(self.hidden_neurons,))

        # 2. ADIM: Gizli Katman Çıktı Matrisinin (H) Hesaplanması
        # H = g(X * w + b)
        linear_model = np.dot(X, self.input_weights) + self.biases
```

```
            H = self._sigmoid(linear_model)

            # 3. ADIM: Çıktı Ağırlıklarının (Beta) Hesaplanması
            # Beta = Moore-Penrose-Inverse(H) * T
            H_pinv = np.linalg.pinv(H)
            self.beta = np.dot(H_pinv, y)

            print(f"Eğitim tamamlandı -> H Matrisi: {H.shape}, Beta Matrisi: {self.be

        def predict(self, X):
            # Tahmin: Y = H * Beta
            linear_model = np.dot(X, self.input_weights) + self.biases
            H = self._sigmoid(linear_model)
            return np.dot(H, self.beta)

    # --- Kodun Çalıştırılması ---

    if __name__ == "__main__":
        # Veri Yükleme
        print("Iris veri seti yükleniyor...")
        iris = load_iris()
        X = iris.data
        y = iris.target.reshape(-1, 1)

        # Normalizasyon (Sigmoid fonksiyonu için kritik)
        scaler = MinMaxScaler()
        X_scaled = scaler.fit_transform(X)

        # One-Hot Encoding
        encoder = OneHotEncoder(sparse_output=False)
        y_encoded = encoder.fit_transform(y)

        # Eğitim/Test Ayrımı
        X_train, X_test, y_train, y_test = train_test_split(
            X_scaled, y_encoded, test_size=0.3, random_state=42, stratify=y
        )

        # Model Eğitimi (20 gizli nöron ile)
        elm = ELMClassifier(hidden_neurons=20)
        print("\nModel eğitiliyor (Tek adımda)...")
        elm.fit(X_train, y_train)

        # Test
        predictions = elm.predict(X_test)

        # Sonuçlar
        y_test_labels = np.argmax(y_test, axis=1)
        y_pred_labels = np.argmax(predictions, axis=1)

        print(f"\nElde Edilen Doğruluk: %{accuracy_score(y_test_labels, y_pred_labels
        print("\nDetaylı Rapor:")
        print(classification_report(y_test_labels, y_pred_labels, target_names=iris.t
```

5. Sonuçların Değerlendirilmesi

Kodu çalıştırdığımda elde ettiğim çıktılar şu şekildedir:

```
Iris veri seti yükleniyor...

Model eğitiliyor (Tek adımda)...
Eğitim tamamlandı -> H Matrisi: (105, 20), Beta Matrisi: (20, 3)

Elde Edilen Doğruluk: %97.78

Detaylı Rapor:
              precision    recall  f1-score   support
      setosa       1.00      1.00      1.00        15
  versicolor       1.00      0.93      0.97        15
   virginica       0.94      1.00      0.97        15
    accuracy                           0.98        45
```

Sonuçlar incelendiğinde;

1. **Matris Boyutları:** 105 eğitim verisi ve 20 gizli nöron kullanıldığı için $\mathbf{H}$ matrisi beklendiği gibi `(105, 20)` boyutundadır.
2. **Yüksek Doğruluk:** Rastgele ağırlık atamasına rağmen, analitik çözüm sayesinde model test verisinde **%97.78** başarı oranına ulaşmıştır.
3. **İşlem Hızı:** Geleneksel yöntemlerin aksine, bu doğruluk oranına yinelemesiz, tek bir