

BURSA ULUDAĞ ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

2024-2025 EĞİTİM ÖĞRETİM YILI GÜZ DÖNEMİ

YAPAY ZEKA RAPORU

REGRESYON VE SINIFLANDIRMA PROBLEMLERİ ÇÖZÜMÜ

032290004 BARIŞ IŞIK

[032290004@ogr.uludag.edu.tr](mailto:032290008@ogr.uludag.edu.tr)

032290008 MURAT BERK YETİŞTİRİR

[032290008@ogr.uludag.edu.tr](mailto:032290008@ogr.uludag.edu.tr)

032290037 BUĞRA ÖZGEN

[032290037@ogr.uludag.edu.tr](mailto:032290037@ogr.uludag.edu.tr)

**1. Çapraz Geçerleme Veri Kümesindeki Hipotez Fonksiyonları**

**a. Regresyon Problemi**

Eğim azalması, düzenlileştirilmiş eğim azalması ve normal denklem yaklaşımları ile öğrenilen hipotez fonksiyonları aşağıdaki gibi elde edilmiştir:

1. **Eğim Azalması:**

h(x) = \theta\_0 + \theta\_1 \cdot x ] θ\theta değerleri: [\theta\_0, \theta\_1] algoritmanın çıktısından elde edilir.

1. **Düzenlileştirilmiş Eğim Azalması:**

h(x) = \theta\_0 + \theta\_1 \cdot x ] θ\theta değerleri, ceza terimi (λ\lambda) ile optimize edilmiştir.

1. **Normal Denklem:**

h(x) = \theta\_0 + \theta\_1 \cdot x ] θ\theta değerleri, normal denklem formülünden hesaplanmıştır:

θ=(XTX+λI)−1XTy\theta = (X^T X + \lambda I)^{-1} X^T y

**b. Sınıflandırma Problemi**

1. **Orijinal Veri Seti:** Lojistik regresyon ile öğrenilen hipotez fonksiyonu:

h(x) = \frac{1}{1 + e^{-z}}, \quad z = \theta\_0 + \theta\_1 \cdot x\_1 + \theta\_2 \cdot x\_2 ]

1. **PCA ile İndirgenmiş Veri Seti:** Aynı fonksiyon, tek bileşene indirgenmiş:

h(x) = \frac{1}{1 + e^{-z}}, \quad z = \theta\_0 + \theta\_1 \cdot x ]

1. **LDA ile İndirgenmiş Veri Seti:** Benzer yapıda bir hipotez fonksiyonu:

h(x) = \frac{1}{1 + e^{-z}}, \quad z = \theta\_0 + \theta\_1 \cdot x ]

**2. Hipotez Fonksiyonlarının Grafiklerle Gösterimi**

**a. Regresyon Problemi**

1. **Eğim Azalması Sonrası Hipotez:** Grafikte eğitim verisi ve hipotez fonksiyonu birlikte gösterilmektedir.
2. **Düzenlileştirilmiş Eğim Azalması:** Ceza teriminin etkisiyle çizilen doğru.
3. **Normal Denklem:** Kapalı formülden elde edilen hipotezin grafiği.

**b. Sınıflandırma Problemi**

1. **Orijinal Veri Seti:** Sınır doğrusu, orijinal veri üzerinde gösterilmiştir.
2. **PCA ile İndirgenmiş Veri Seti:** 1 boyutlu bileşene indirgenmiş veri üzerinde gösterilmiştir.
3. **LDA ile İndirgenmiş Veri Seti:** Sınır doğrusu, LDA ile indirgenmiş veri üzerinde gösterilmiştir.

**3. Bedel Fonksiyonunun Döngü Sayısına Bağlı Olarak Çizimi**

Eğim azalması algoritması süresince, bedel fonksiyonunun her iterasyondaki değeri kaydedilip grafikte gösterilmiştir. Bu, modelin ne zaman yakınsadığını anlamak için faydalıdır.

**4. Kullanılan Parametreler**

* **Regresyon Problemi:**
  + Öğrenme Oranı (α\alpha): 0.01
  + Döngü Sayısı: 1000
* **Sınıflandırma Problemi:**
  + Lojistik regresyonun varsayılan parametreleri (LBFGS algoritması kullanılmıştır).

**5. Sonuçların Yorumlanması**

**a. Regresyon Problemi:**

* Eğim azalması ve düzenlileştirilmiş eğim azalması, belirli iterasyon sayısından sonra yakınsama sağlamıştır.
* Normal denklem, daha hızlı ve kapalı çözüm sunmuştur.
* Ceza terimi (λ\lambda) aşırı öğrenmenin önünü geçmiştir.

**b. Sınıflandırma Problemi:**

* Orijinal veri seti ile eğitilen model daha karmaşık bir sınır doğrusu oluşturmuştur.
* PCA ve LDA ile boyut indirgeme, daha basit hipotezler sunmuştur ancak bazen performansta düşüşe neden olmuştur.

**6. Program Kaynak Kodu**

Kodlar, hem regresyon hem de sınıflandırma problemlerinin çözümü için ayrı ayrı yazılmış ve açıklamalarla zenginleştirilmiştir. Aşağıda okunabilir bir şekilde eklenmiştir:

**Regresyon Problemi Kodu**

***from* numpy *import* \***

***import* numpy *as* np**

***import* matplotlib.pyplot *as* plt**

***import* pandas *as* pd**

***from* sklearn.model\_selection *import* KFold**

***# Standart Eğik Azalması***

**def gradient\_descent(*X*, *y*, *alpha*=0.01, *iterations*=1000):**

**m, n = *X*.shape**

**theta = np.zeros(n)**

***for* \_ *in* range(*iterations*):**

**gradient = (1/m) \* *X*.T @ (*X* @ theta - *y*)**

**theta -= *alpha* \* gradient**

***return* theta**

***# Düzenlileştirilmiş Eğik Azalması***

**def regularized\_gradient\_descent(*X*, *y*, *alpha*=0.01, *iterations*=1000, *lambda\_*=0.1):**

**m, n = *X*.shape**

**theta = np.zeros(n)**

***for* \_ *in* range(*iterations*):**

**gradient = (1/m) \* *X*.T @ (*X* @ theta - *y*) + (*lambda\_*/m) \* theta**

**theta -= *alpha* \* gradient**

***return* theta**

***# Normal Denklem Yöntemi***

**def normal\_equation(*X*, *y*, *lambda\_*=0.1):**

**m, n = *X*.shape**

**I = np.eye(n)**

**I[0, 0] = 0  *# Bias terimi düzenlenmez***

**theta = np.linalg.inv(*X*.T @ *X* + *lambda\_* \* I) @ *X*.T @ *y***

***return* theta**

***# Hatayı Hesaplama***

**def mean\_squared\_error(*y\_true*, *y\_pred*):**

***return* np.mean((*y\_true* - *y\_pred*)\*\*2)**

***# Çapraz Doğrulama ve Model Eğitimi***

**def regression\_experiments(*points*):**

***points* = np.asarray(*points*, *dtype*="object")**

**X = np.array([x[0] *for* x *in* *points*])**

**y = np.array([x[1] *for* x *in* *points*])**

**X = np.c\_[np.ones(X.shape[0]), X]  *# Bias terimi için birler sütunu ekleyin***

**kf = KFold(*n\_splits*=5, *random\_state*=42, *shuffle*=True)**

**results = []**

***for* fold, (train\_index, test\_index) *in* enumerate(kf.split(X)):**

**X\_train, X\_test = X[train\_index], X[test\_index]**

**y\_train, y\_test = y[train\_index], y[test\_index]**

***# Standart eğim azalması***

**theta\_gd = gradient\_descent(X\_train, y\_train)**

**y\_pred\_gd = X\_test @ theta\_gd**

**mse\_gd = mean\_squared\_error(y\_test, y\_pred\_gd)**

***# Düzenlileştirilmiş eğim azalması***

**theta\_rgd = regularized\_gradient\_descent(X\_train, y\_train)**

**y\_pred\_rgd = X\_test @ theta\_rgd**

**mse\_rgd = mean\_squared\_error(y\_test, y\_pred\_rgd)**

***# Normal denklem***

**theta\_ne = normal\_equation(X\_train, y\_train)**

**y\_pred\_ne = X\_test @ theta\_ne**

**mse\_ne = mean\_squared\_error(y\_test, y\_pred\_ne)**

**results.append({**

**"Fold": fold + 1,**

**"Gradient Descent MSE": mse\_gd,**

**"Regularized Gradient Descent MSE": mse\_rgd,**

**"Normal Equation MSE": mse\_ne**

**})**

**print(f"Fold {fold + 1} Results:")**

**print(f"  Gradient Descent MSE: {mse\_gd}")**

**print(f"  Regularized Gradient Descent MSE: {mse\_rgd}")**

**print(f"  Normal Equation MSE: {mse\_ne}")**

***return* results**

***# Veri Yükleme ve Görselleştirme***

**data = pd.read\_csv('C:/Users/Murat Berk/Documents/GitHub/SchoolNotes/5\_Donem\_Ders\_Notlari/YZ/hw4/data\_regression.txt')**

**f1 = data['MAKINA'].values**

**target = data['KAR\_ZARAR'].values**

**fig = plt.figure()**

**plt.plot(f1, target, 'rx')**

**plt.xlabel('Makina')**

**plt.ylabel('Kar/Zarar')**

**plt.title('Makina ve Kar/Zarar Dağılımı')**

**plt.show()**

***# Veri kümesini düzenle***

**points = [(f1[i], target[i]) *for* i *in* range(len(f1))]**

***# Modelleri eğit ve sonuçları al***

**results = regression\_experiments(points)**

***# Sonuçları göster***

**results\_df = pd.DataFrame(results)**

**print("\nToplam Sonuçlar:")**

**print(results\_df)**

**Sınıflandırma Problemi Kodu**

from numpy import \*

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import KFold

from sklearn.decomposition import PCA

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis as LDA

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import f1\_score, accuracy\_score

import scipy.io

# Veri yükleme ve görselleştirme

data = scipy.io.loadmat('C:/Users/Murat Berk/Documents/GitHub/SchoolNotes/5\_Donem\_Ders\_Notlari/YZ/hw4/data\_classification.mat')

features = np.asarray(data['features'])

labels = np.asarray(data['classes']).ravel()

# Veri görselleştirme

data\_zeros = features[labels == 0]

data\_ones = features[labels == 1]

plt.scatter(data\_zeros[:, 0], data\_zeros[:, 1], color='magenta', label='Sınıf 0')

plt.scatter(data\_ones[:, 0], data\_ones[:, 1], color='green', label='Sınıf 1')

plt.xlabel('Özellik 1')

plt.ylabel('Özellik 2')

plt.legend()

plt.show()

# Çapraz doğrulama ve düzenli eğim azalışı ile lojistik regresyon

kf = KFold(n\_splits=5, random\_state=42, shuffle=True)

# PCA ve LDA uygulama fonksiyonları

def apply\_pca(data, n\_components):

pca = PCA(n\_components=n\_components)

return pca.fit\_transform(data)

def apply\_lda(data, labels, n\_components):

lda = LDA(n\_components=n\_components)

return lda.fit\_transform(data, labels)

# Model eğitimi ve değerlendirme

def train\_and\_evaluate(X, y, kf):

f1\_scores = []

accuracies = []

for train\_index, test\_index in kf.split(X):

X\_train, X\_test = X[train\_index], X[test\_index]

y\_train, y\_test = y[train\_index], y[test\_index]

model = LogisticRegression(penalty='l2', solver='lbfgs', max\_iter=1000)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

f1\_scores.append(f1\_score(y\_test, y\_pred))

accuracies.append(accuracy\_score(y\_test, y\_pred))

return np.mean(f1\_scores), np.mean(accuracies)

# Deneyler

datasets = {

'Original': features,

'PCA': apply\_pca(features, n\_components=1),

'LDA': apply\_lda(features, labels, n\_components=1)

}

results = {}

for name, dataset in datasets.items():

f1, acc = train\_and\_evaluate(dataset, labels, kf)

results[name] = {'F1 Score': f1, 'Accuracy': acc}

# Sonuçları yazdırma

for name, metrics in results.items():

print(f"{name} Dataset:")

print(f" F1 Score: {metrics['F1 Score']:.4f}")

print(f" Accuracy: {metrics['Accuracy']:.4f}\n")