KNN İle Sınıflandırma ve Regresyon Modeli

Alican Tunç
Veri Bilimi ve Büyük Veri Yüksek Lisans Programı
Yıldız Teknik Üniversitesi
Istanbul, TÜRKİYE:
can.tunc1@std.yildiz.edu.tr

***Abstract*—Bu çalışmada, sınıflandırma problemlerinde K-En Yakın Komşu (KNN) algoritması adım adım uygulanarak performansı değerlendirilmiştir. Veri seti %70 eğitim ve %30 test olarak ikiye ayrılmış, Öklidyen mesafe kullanılarak veri noktaları arasındaki benzerlikler hesaplanmıştır. KNN algoritması elle kodlanarak, belirli bir sorgu noktası için en yakın K komşusu tespit edilmiş ve sınıflandırma işlemi çoğunluk oyu yöntemiyle gerçekleştirilmiştir. Farklı K değerleri için modelin doğruluğu hesaplanmış ve en iyi K değeri grafikle analiz edilmiştir. Ayrıca, çoklu doğrusal regresyon modeli kurularak fiyat, bölge ve ülke değişkenlerinin satışlar üzerindeki etkileri değerlendirilmiş, istatistiksel anlamlılık testleri yapılmış ve sade bir model önerilmiştir. Elde edilen bulgular, hem sınıflandırma hem de regresyon açısından temel istatistiksel yorumlamalar sunmaktadır.**

Keywords—K-En yakın komşu, regresyon modeli, sınıflandırma, doğruluk analizi,

I. GIRIŞ

Makine öğrenmesi ve istatistiksel modelleme, veri analitiği alanında güçlü araçlardır. Bu çalışmada, iki temel yöntem olan K-En Yakın Komşu (KNN) algoritması ve çoklu doğrusal regresyon modelleri incelenmiştir. KNN algoritması, sınıflandırma problemlerinde sıkça kullanılan sezgisel ve etkili bir yöntemdir. Bu algoritma, veri noktaları arasındaki mesafeye dayanarak sınıflandırma yapar ve karar verme sürecinde en yakın komşuların etiketlerini dikkate alır.

Çalışmanın ilk aşamasında, KNN algoritması elle kodlanmış ve farklı K değerleri için model başarımı test edilmiştir. Ardından, çoklu doğrusal regresyon modeli kurularak "Sales" (Satışlar) değişkeni hedef olarak alınmış; Price, Urban ve US değişkenlerinin satışlar üzerindeki etkileri incelenmiştir. Regresyon modeli kapsamında katsayıların anlamları yorumlanmış, anlamlı değişkenler ile daha sade bir model önerilmiş ve modeller R-kare değerleri ile karşılaştırılmıştır. Böylece, hem sınıflandırma hem de regresyon bağlamında kapsamlı bir analiz gerçekleştirilmiştir. Bunu yaparken python ve R programlama dilleri kullanılmıştır.

II. DENEYSEL ANALİZ

A. Veri Seti

Bu çalışmada üç farklı veri seti kullanılmıştır: ***Iris****,* ***Carseats*** *ve* ***Movielens 100k****.*

Iris veri seti*, çiçek türlerini sınıflandırmak için sıklıkla kullanılan klasik bir veri setidir. Her bir örnek dört sayısal özelliğe (sepallength, sepalwidth, petallength, petalwidth) sahiptir ve hedef değişken olarak çiçeğin türü (****Iris Setosa****,* ***Iris Versicolour*** *veya* ***Iris Virginica****) belirlenmiştir. Bu veri seti, çok sınıflı sınıflandırma problemleri için uygundur ve algoritmaların temel performanslarını karşılaştırmak amacıyla kullanılmıştır.*

Carseats veri seti*, farklı mağazalardaki çocuk oto koltuğu satışlarını etkileyen faktörleri içermektedir. Hedef değişken olarak* ***Sales*** *(Satışlar) kullanılmış; açıklayıcı değişkenler arasında* ***Price*** *(Fiyat),* ***Urban*** *(mağazanın kentsel bölgede olup olmaması),* ***US*** *(ABD'de bulunma durumu) gibi sayısal ve kategorik öznitelikler yer almıştır. Bu veri seti üzerinden* ***çoklu doğrusal regresyon analizi*** *gerçekleştirilmiştir.*

Movielens 100k veri seti *ise, kullanıcıların filmlere verdiği puanları içermektedir ve* ***öneri sistemleri*** *bağlamında analiz edilmiştir. Veri seti; kullanıcı kimliği, film kimliği, kullanıcı puanı ve zaman damgası gibi bilgileri içermektedir. Bu veri üzerinden kullanıcıların beğeni kalıpları analiz edilerek modelleme çalışmaları yapılmıştır.*

Bu üç veri seti, farklı problem türlerini (sınıflandırma, regresyon ve öneri sistemleri) temsil etmeleri açısından seçilmiş ve ilgili algoritmaların bu problemlerdeki performansları değerlendirilmiştir.

B. *Yöntem*

Bu çalışmada, hem KNN hem de regreson modelleri aşağıdaki şekilde eğitilmiştir:

C. 3.1 KNN:

K-en Yakın Komşu (KNN) yöntemi, örneklerin sınıf etiketlerini komşularına bakarak belirleyen denetimli bir öğrenme algoritmasıdır. Bu çalışmada KNN algoritması farklı ***K*** *değerleri (komşu sayısı) denenerek uygulanmıştır. Modelin başarımını artırmak amacıyla hiperparametre optimizasyonu yapılmış; en uygun* ***K*** *değeri, doğrulama veri seti üzerinde elde edilen* ***doğruluk (accuracy)*** *değerlerine göre belirlenmiştir. Farklı K değerleri için model eğitilerek doğrulama setinde elde edilen doğruluklar karşılaştırılmış ve en iyi sonucu veren K değeri seçilmiştir. Ayrıca, modelin karar sınırları grafiksel olarak görselleştirilerek KNN'nin veri kümesi üzerindeki sınıflandırma başarısı detaylı şekilde analiz edilmiştir.*

C. *3.2 Çoklu Doğrusal Regresyon Modeli:*

Bu çalışmada, Carseats veri seti kullanılarak ***Satışlar (Sales)*** *değişkeni hedef (bağımlı) değişken olarak belirlenmiş ve bu değişkeni etkileyebileceği düşünülen bazı bağımsız değişkenlerle* ***çoklu doğrusal regresyon*** *modeli kurulmuştur. İlk modelde açıklayıcı değişken olarak* ***Price (Fiyat), Urban (Kentsel/Bölge)*** *ve* ***US (ABD'de olup olmama durumu)*** *değişkenleri kullanılmıştır. Urban ve US değişkenleri kategorik yapıda olduğu için modelde referans kategorilere göre dummy (sıfır-bir) kodlama yöntemiyle dahil edilmiştir.*

Model sonucunda elde edilen katsayılar ve p-değerleri kullanılarak her bir değişkenin ***istatistiksel olarak anlamlılığı*** *test edilmiştir. Bu testler sonucunda anlamlı bulunan değişkenler belirlenmiş ve yalnızca bu değişkenlerle* ***sadeleştirilmiş bir model*** *oluşturulmuştur. İlk ve sadeleştirilmiş modellerin başarımları,* ***R-kare (R^2)*** *ve* ***Düzeltilmiş R-kare (Adjusted R^2)*** *değerleri ile karşılaştırılmış, modelin açıklayıcılığı analiz edilmiştir.*

Son olarak, sade modelin katsayıları için ***%95 güven aralıkları*** *hesaplanmış; modelde aykırı gözlemler veya yüksek etki yaratan noktalar olup olmadığı grafiksel ve istatistiksel yöntemlerle incelenmiştir. Bu analizlerde* ***etki grafikleri*** *ve* ***Cook'un uzaklığı*** *gibi yöntemlerden yararlanılmıştır.*

D. *3.2.1 KNN için İşlemler*

Knn, için python kullanarak aşağıdaki şekilde öklidyen mesafesine göre tanımlayıp modelimizi kuruyoruz.

```python
# Öklidyen mesafe
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))

# En yakın k komşuyu bul
def get_k_neighbors(X_train, y_train, test_point, k):
    distances = []
    for i in range(len(X_train)):
        dist = euclidean_distance(test_point, X_train[i])
        distances.append((dist, y_train[i]))
    distances.sort(key=lambda x: x[0])
    neighbors = distances[:k]
    return [label for _, label in neighbors]

# Tahmin fonksiyonları
def predict_point(X_train, y_train, test_point, k):
    neighbors = get_k_neighbors(X_train, y_train, test_point, k)
    return Counter(neighbors).most_common(1)[0][0]

def predict_knn(X_train, y_train, X_test, k):
    return np.array([predict_point(X_train, y_train, x, k) for x in X_test])
```

Şekil 1: KNN modelinin oluşturulması

```python
# En iyi k'yi bul (test seti üzerinden)
def find_best_k_on_test(X_train, y_train, X_test, y_test, k_values=None):
    if k_values is None:
        k_values = list(range(1, 51))

    test_accuracies = []
    for k in k_values:
        y_pred = predict_knn(X_train, y_train, X_test, k)
        acc = accuracy_score(y_test, y_pred)
        test_accuracies.append(acc)

    best_k = k_values[np.argmax(test_accuracies)]
    print(f"\nEn iyi k: {best_k} | Test Doğruluğu: {max(test_accuracies):.2f}")

    # Grafik
    plt.figure(figsize=(10, 5))
    plt.plot(k_values, test_accuracies, marker='o', linestyle='--', color='green')
    plt.xlabel("k Değeri")
    plt.ylabel("Test Doğruluğu")
    plt.title("Test Doğruluğu vs K Değeri")
    plt.grid(True)
    plt.show()

    return best_k
```

Şekil 2: Knn modelinin eğitimi ve best k paramatresi

Sonrasında modelimizi eğitmek için train fonksiyonu ve eğitimdeki çıktıları görmek için acc değerlerini görmek için ayrıca bir fonksiyon kullanacağız. Bunun için kları döndürecek bir for dongüsü kullanıyoruz.

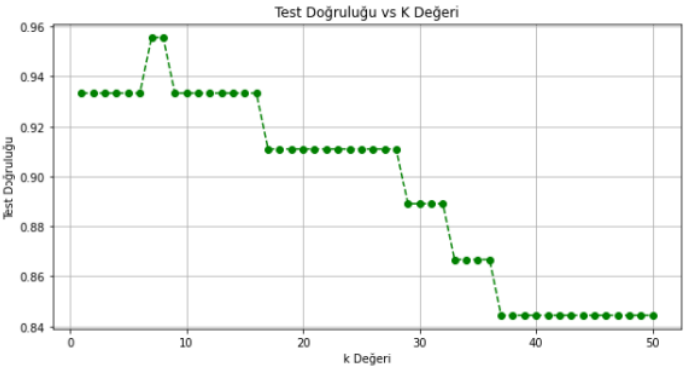
```python
# Performans değerlendirme
def evaluate_performance(X_train, y_train, X_test, y_test, k, class_names=None):
    y_pred = predict_knn(X_train, y_train, X_test, k)
    acc = accuracy_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)

    print(f"\nK = {k} için Test Doğruluğu: {acc:.2f}")
    print("\nClassification Report:\n")
    print(classification_report(y_test, y_pred, target_names=class_names))

    plt.figure(figsize=(6, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=class_names, yticklabels=class_names)
    plt.xlabel("Tahmin")
    plt.ylabel("Gerçek")
    plt.title("Confusion Matrix")
    plt.show()
    return acc
```

Şekil 3: Acc hesabı

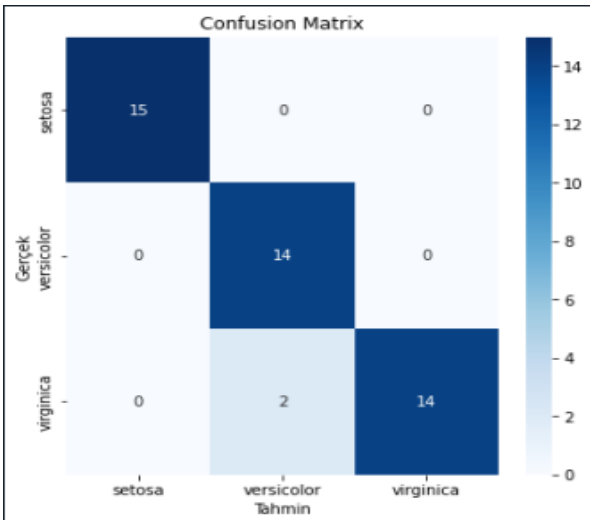
```python
iris = load_iris()
X = iris.data
y = iris.target
class_names = iris.target_names

# %70 eğitim, %30 test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=450)

# En iyi k ve değerlendirme
best_k = find_best_k_on_test(X_train, y_train, X_test, y_test)
evaluate_performance(X_train, y_train, X_test, y_test, best_k, class_names=class_names)
```

Şekil 4: Veri seti

İlk olarak başlangıçtaki parametreleri bilmediğimiz 1-50 kadar k değeri vererek eğitime başlıyoruz. Modeli eğitip acc değerlerini tüm k değerleri için bakacak olursak aşağıdaki sonuçları elde edeceğiz.



Şekil 5: Tüm k değerleri vs acc

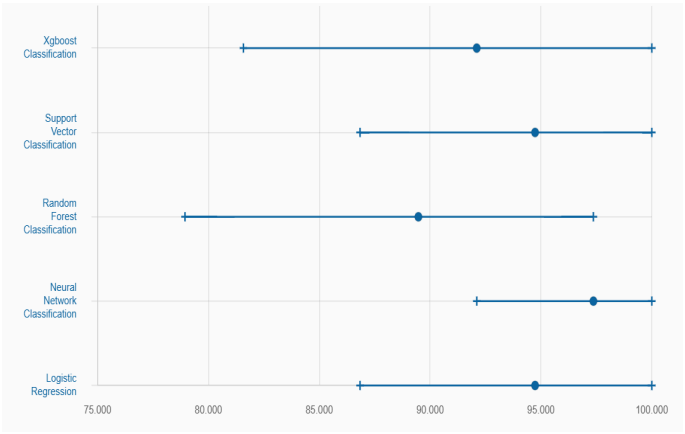


Şekil 6: Conf matrix

```
En iyi k: 7 | Test Doğruluğu: 0.96

K = 7 için Test Doğruluğu: 0.96

Classification Report:

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.88      1.00      0.93        14
   virginica       1.00      0.88      0.93        16

    accuracy                           0.96        45
   macro avg       0.96      0.96      0.96        45
weighted avg       0.96      0.96      0.96        45
```

Şekil 7: En iyi k değeri için çıktılar

Şekil 5'deki grafiğe bakacak olursak en yüksek doğruluk değerinin k=7 olduğu zaman olduğunu görebiliriz ayrıca yine Şekil 6-7 ilgili çıktının detaylı analizi görebiliriz. Küçük ve önceden hazırlanmış bir data seti olduğu için bu kadar yüksek bir acc değeri elde etmemiz çok normal ve https://archive.ics.uci.edu/dataset/53/iris sitesindeki veriyle uyuşmakta.



Şekil 8: İlgili data setin farklı modellere göre acc değerleri

E. *3.2.2 KNN Tabanlı Tavsiye Sistemleri: User-KNN ve Item-KNN Karşılaştırması*

*Bu çalışmada, **Movielens 100k** veri seti kullanılarak kullanıcıların film değerlendirmelerine dayalı **tavsiye sistemleri** geliştirilmiştir. Kullanıcıların film değerlendirmelerini temel alan iki farklı yaklaşım uygulanmıştır: **Kullanıcı-Temelli KNN (User-KNN)** ve **Öğe-Temelli KNN (Item-KNN)**.*

*Veri seti öncelikle eğitim ve test kümelerine ayrılmıştır. **%80 eğitim** ve **%20 test oranı** ile ayrılan veri setinde, kullanıcı-öğe etkileşimleri pivot tablo şeklinde yapılandırılmıştır. Eksik puanlar `0` ile doldurularak **cosine benzerliği** kullanımı için uygun hale getirilmiştir. Benzerlik matrisi hem kullanıcılar hem de öğeler (filmler) için ayrı ayrı hesaplanmıştır.*

Kullanıcı-Temelli KNN (User-KNN):

Bu yöntem, bir kullanıcının ilgilenebileceği öğeleri, benzer değerlendirme profiline sahip diğer kullanıcıların tercihlerine dayanarak tahmin eder. Her kullanıcı için en benzer K kullanıcı belirlenmiş ve bu kullanıcıların hedef öğeye verdikleri puanların ağırlıklı ortalaması alınarak tahmin yapılmıştır.

Öğe-Temelli KNN (Item-KNN):

Bu yöntemde ise tahminler, kullanıcının daha önce puanladığı öğelere benzer filmler üzerinden yapılmıştır. Belirli bir film için en benzer K öğe tespit edilerek, kullanıcının bu benzer öğelere verdiği puanlar kullanılmıştır.

*Her iki yöntem için farklı **K değerleri (5, 10, 20, 30, ..., 150)** denenmiş ve **test kümesinde RMSE (Root Mean Squared Error)** değeri hesaplanmıştır. RMSE, tahmin edilen puanlar ile gerçek puanlar arasındaki farkların karelerinin ortalamasının karekökü alınarak elde edilmiştir.*

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import mean_squared_error

# Dosya yolunu belirtelim
file_path = "C:\\Users\\Alican\\Desktop\\Ödev\\ml-100k\\u.data"

# Veri setini yükleyelim
columns = ['user_id', 'item_id', 'rating', 'timestamp']
data = pd.read_csv(file_path, sep='\t', names=columns)

# Eğitim ve test setlerine ayıralım
train_data, test_data = train_test_split(data, test_size=0.2, random_state=100)

print(f"Eğitim seti boyutu: {train_data.shape}")
print(f"Test seti boyutu: {test_data.shape}")
```

Şekil 9: Veri seti kurulumu

```
Eğitim seti boyutu: (80000, 4)
Test seti boyutu: (20000, 4)
```

Şekil 10:Train ve test veri seti

```python
# Kullanıcı-Öğe matrisi oluştur
user_item_matrix = train_data.pivot(index='user_id', columns='item_id', values='rating').fillna(0)

# Kullanıcılar arası cosine benzerlik matrisi
user_similarity = cosine_similarity(user_item_matrix)
user_similarity_df = pd.DataFrame(user_similarity, index=user_item_matrix.index, columns=user_item_matrix.index)

def predict_user_based(user_id, item_id, k=5):
    # Kullanıcı ve öğe ID'sinin veri setinde bulunup bulunmadığını kontrol et
    if user_id not in user_similarity_df.index or item_id not in user_item_matrix.columns:
        return np.nan

    # Kullanıcının benzerlik skorlarını al
    user_similarities = user_similarity_df[user_id]

    # Kullanıcının benzerlik skorlarına göre sıralama
    similar_users = user_similarities.sort_values(ascending=False).index[1:k+1]

    # Benzer kullanıcıların öğe değerlendirmelerini al
    similar_users_ratings = user_item_matrix.loc[similar_users, item_id]

    # Tahmini hesapla
    prediction = np.dot(user_similarities[similar_users], similar_users_ratings) / np.sum(user_similarities[similar_use
    return prediction

def calculate_rmse_user_based(k):
    test_data['prediction'] = test_data.apply(lambda x: predict_user_based(x['user_id'], x['item_id'], k), axis=1)
    valid_predictions = test_data['prediction'].notna()
    filtered_test_data = test_data[valid_predictions]
    rmse = np.sqrt(mean_squared_error(filtered_test_data['rating'], filtered_test_data['prediction']))
    return rmse
```

Şekil 11:User-knn

```python
# Öğe-temelli KNN (Item-KNN) Algoritması
item_user_matrix = train_data.pivot(index='item_id', columns='user_id', values='rating').fillna(0)

# Öğeler arası cosine benzerlik matrisi
item_similarity = cosine_similarity(item_user_matrix)
item_similarity_df = pd.DataFrame(item_similarity, index=item_user_matrix.index, columns=item_user_matrix.index)

def predict_item_based(user_id, item_id, k=5):
    # Kullanıcı ve öğe ID'sinin veri setinde bulunup bulunmadığını kontrol et
    if item_id not in item_similarity_df.index or user_id not in item_user_matrix.columns:
        return np.nan

    # Öğenin benzerlik skorlarını al
    item_similarities = item_similarity_df[item_id]

    # Öğenin benzerlik skorlarına göre sıralama
    similar_items = item_similarities.sort_values(ascending=False).index[1:k+1]

    # Benzer öğelerin kullanıcı değerlendirmelerini al
    similar_items_ratings = item_user_matrix.loc[similar_items, user_id]

    # Tahmini hesapla
    prediction = np.dot(item_similarities[similar_items], similar_items_ratings) / np.sum(item_similarities[similar_item
    return prediction

def calculate_rmse_item_based(k):
    test_data['prediction'] = test_data.apply(lambda x: predict_item_based(x['user_id'], x['item_id'], k), axis=1)
    valid_predictions = test_data['prediction'].notna()
    filtered_test_data = test_data[valid_predictions]
    rmse = np.sqrt(mean_squared_error(filtered_test_data['rating'], filtered_test_data['prediction']))
    return rmse
```

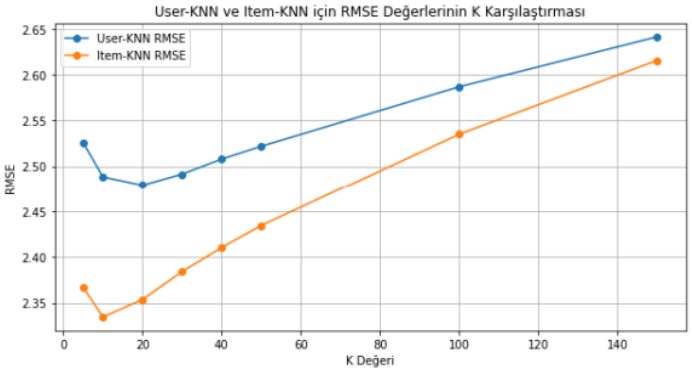
Şekil 12:İtem-knn

```python
# Farklı K değerleri için RMSE hesaplama
k_values = [5, 10, 20, 30, 40, 50, 100, 150]
results = []
user_rmse_values = []
item_rmse_values = []

for k in k_values:
    user_rmse = calculate_rmse_user_based(k)
    item_rmse = calculate_rmse_item_based(k)
    user_rmse_values.append(user_rmse)
    item_rmse_values.append(item_rmse)
    results.append({
        "Yöntem": "User-KNN",
        "Hiperparametreler": f"K = {k}, cosine",
        "Test RMSE": user_rmse
    })
    results.append({
        "Yöntem": "Item-KNN",
        "Hiperparametreler": f"K = {k}, cosine",
        "Test RMSE": item_rmse
    })

# Sonuçları tablo şeklinde sunma
results_df = pd.DataFrame(results)
print(results_df)
```

Şekil 13:Farklı K değerleri için RMSE hesaplama

```python
# Sonuçları tablo şeklinde sunma
results_df = pd.DataFrame(results)
print(results_df)

# RMSE değerlerini plot etme
plt.figure(figsize=(10, 5))
plt.plot(k_values, user_rmse_values, marker='o', label='User-KNN RMSE')
plt.plot(k_values, item_rmse_values, marker='o', label='Item-KNN RMSE')
plt.xlabel('K Değeri')
plt.ylabel('RMSE')
plt.title('User-KNN ve Item-KNN için RMSE Değerlerinin K Karşılaştırması')
plt.legend()
plt.grid()
plt.show()
```

Şekil 14:Sonuç çıktılarını veren kod satırı



Şekil 15: K değerlerinin plot hali

	Yöntem	Hiperparametreler	Test RMSE
0	User-KNN	K = 5, cosine	2.525718
1	Item-KNN	K = 5, cosine	2.366771
2	User-KNN	K = 10, cosine	2.488345
3	Item-KNN	K = 10, cosine	2.334657
4	User-KNN	K = 20, cosine	2.479184
5	Item-KNN	K = 20, cosine	2.353714
6	User-KNN	K = 30, cosine	2.491504
7	Item-KNN	K = 30, cosine	2.384197
8	User-KNN	K = 40, cosine	2.508037
9	Item-KNN	K = 40, cosine	2.410356
10	User-KNN	K = 50, cosine	2.521956
11	Item-KNN	K = 50, cosine	2.434455

Şekil 16: Farklı K değerleri için RMSE çıktıları

*Elde edilen sonuçlara göre, her bir K değeri için **Item-KNN yönteminin RMSE değeri genellikle daha düşük** olmuş, yani daha başarılı tahminler yapabilmiştir. Ayrıca her iki yöntemde de K değeri arttıkça, RMSE değerinin kademeli olarak yükseldiği gözlemlenmiştir. Bu durum, daha az benzer kullanıcı veya öğenin daha etkili tahminler ürettiğini ve yüksek K değerlerinde tahminlerin genelleştirildiğini göstermektedir.*

Sonuç ve Değerlendirme:

*Bu çalışma, KNN temelli tavsiye sistemlerinin uygulanabilirliğini ve farklı yapılandırmaların model başarısına etkisini ortaya koymuştur. Özellikle **Item-KNN algoritması**, Movielens veri seti üzerinde daha düşük hata oranı ile daha iyi performans sergilemiştir. Bu durum, kullanıcıların film tercihlerinin benzer içeriklere dayalı olarak daha isabetli modellenebileceğini göstermektedir. Ayrıca model eğitilirken surprise, lightfm, implicit gibi kütüphaneler uyumsuzluk(bilgisayarın) nedeniyle kullanılmamıştır.*

F. 3.3 Çoklu Doğrusal Regresyon Modeli

*Carseats veri seti kullanılarak, bağımlı değişken olarak **Sales** ve bağımsız değişkenler olarak **Price**, **Urban**, ve **US** seçilmiştir. Bu bağlamda oluşturulan çoklu doğrusal regresyon modeli aşağıdaki şekilde tanımlanmıştır:*

```
Call:
lm(formula = Sales ~ Price + Urban + US, data = Carseats)

Residuals:
    Min      1Q  Median      3Q     Max
-6.9206 -1.6220 -0.0564  1.5786  7.0581

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 13.043469   0.651012  20.036  < 2e-16 ***
Price       -0.054459   0.005242 -10.389  < 2e-16 ***
UrbanYes    -0.021916   0.271650  -0.081    0.936
USYes        1.200573   0.259042   4.635 4.86e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.472 on 396 degrees of freedom
Multiple R-squared:  0.2393,    Adjusted R-squared:  0.2335
F-statistic: 41.52 on 3 and 396 DF,  p-value: < 2.2e-16
```

Şekil 17: Çoklu doğrusal model

Modelde tahmin edilen katsayılar ve yorumları:

- ***Intercept (β_0 = 13.04):** Tüm bağımsız değişkenler sabitken, yani Price = 0, Urban = No, US = No olduğunda, satışların ortalama değeri 13.04 birimdir.*
- ***Price (β_1 = -0.054):** Fiyat değişkeninde meydana gelen bir birimlik artış, satışlarda ortalama 0.054 birimlik azalmaya neden olmaktadır. Bu sonuç, satışlar üzerinde negatif ve anlamlı bir fiyata duyarlılık olduğunu gösterir.*
- ***Urban[Yes] (β_2 = -0.022):** Satış noktasının kentsel bölgede bulunması, satışlara anlamlı bir etki yapmamaktadır. p-değeri oldukça yüksek olup (%93.6), istatistiksel olarak anlamlı değildir.*
- ***US[Yes] (β_3 = 1.201):** Satış noktasının ABD'de bulunması, satışları ortalama 1.2 birim artırmaktadır. Bu etki istatistiksel olarak anlamlıdır ($p < 0.01$).*

Not: Urban ve US değişkenleri ikili (binary) kategorik değişkenler olup, referans kategoriler sırasıyla "No" olarak alınmıştır.

```
Model Denklem Formu:
Sales = β0 + β1 * Price + β2 * Urban[T.Y] + β3 * US[T.Y] + ε
```

Şekil 18: Çoklu doğrusal modelin denklem formu

Burada;

- `Urban[Yes]`*: Urban değişkeni "Yes" ise 1, aksi halde 0.*
- `US[Yes]`*: US değişkeni "Yes" ise 1, aksi halde 0.*

*Bu şekilde, kategorik değişkenler **dummy (kukla) değişkenler** olarak modele dahil edilmiştir.*

```
Sıfır Hipotezinin Testi (p-değerleri):
> summary(model)$coefficients[,4]  # p-değerlerini görüntüleyelim
 (Intercept)        Price     UrbanYes        USYes
3.626602e-62 1.609917e-22 9.357389e-01 4.860245e-06
> # (e) Anlamlı bulunan değişkenleri kullanarak sadeleştirilmiş model
> # Burada, p-değerlerine göre anlamlı olan değişkenleri seçiyoruz
> model_sade <- lm(Sales ~ Price + US, data = Carseats)
> # (f) İki modelin uyumunu karşılaştıralım
> cat("\nİki Modelin Uyum Karşılaştırması: \n")
```

Şekil 19: Hipotez ve iki modelin uyumu

*Modelde yer alan her bir bağımsız değişken için sıfır hipotezi (**H_0: β = 0**) test edilmiştir. Değişkenlerin anlamlılıkları aşağıdaki gibidir:*

Değişken	***p-değeri***	***Anlamlılık Durumu***
Price	*<0.001*	*Anlamlı ✅*
Urban[Yes]	*0.936*	*Anlamlı değil ❌*
US[Yes]	*<0.001*	*Anlamlı ✅*

*Sonuç olarak, **Price** ve **US** değişkenleri satışlar üzerinde istatistiksel olarak anlamlı etkiye sahiptir. **Urban** değişkeni ise anlamlı bulunmamıştır. Anlamlılığı olmayan **Urban** değişkeni modelden çıkarılarak, yalnızca anlamlı bulunan bağımsız değişkenler (Price ve US) ile yeni bir model oluşturulmuştur. Tahmin edilen sade model katsayıları, önceki modeliyle oldukça benzerdir ve bu sade model daha az parametreyle benzer açıklayıcılığa sahiptir.*

```
İki Modelin Uyum Karşılaştırması:
> cat("İlk Model R²: ", summary(model)$r.squared, "\n")
İlk Model R²:  0.2392754
> cat("İlk Model Düzeltilmiş R²: ", summary(model)$adj.r.squared, "\n")
İlk Model Düzeltilmiş R²:  0.2335123
> cat("Sadeleştirilmiş Model R²: ", summary(model_sade)$r.squared, "\n")
Sadeleştirilmiş Model R²:  0.2392629
> cat("Sadeleştirilmiş Model Düzeltilmiş R²: ", summary(model_sade)$adj.r.squared, "\n")
Sadeleştirilmiş Model Düzeltilmiş R²:  0.2354305
> # (g) Sadeleştirilmiş modelde katsayılar için %95 güven aralıkları
> cat("\nSadeleştirilmiş Modelin Katsayıları için %95 Güven Aralıkları: \n")
```

Şekil 20: Çıktılar

*Her iki modelin açıklayıcılığı **R-kare** ve **Düzeltilmiş R-kare** değerleriyle karşılaştırılmıştır:*

Model	***R^2***	***Adj. R^2***
Tam Model	*0.239*	*0.234*
Sadeleştirilmiş	*0.239*	*0.235*

Sonuçlara göre sade model, daha az değişkenle benzer düzeyde açıklayıcılığa sahiptir. Urban değişkeninin çıkarılması model performansında kayda değer bir azalma yaratmamıştır.

```
Sadeleştirilmiş Modelin Katsayıları için %95 Güven Aralıkları:
> confint(model_sade, level = 0.95)
                2.5 %     97.5 %
(Intercept) 11.79032020 14.27126531
Price       -0.06475984 -0.04419543
USYes        0.69151957  1.70776632
```

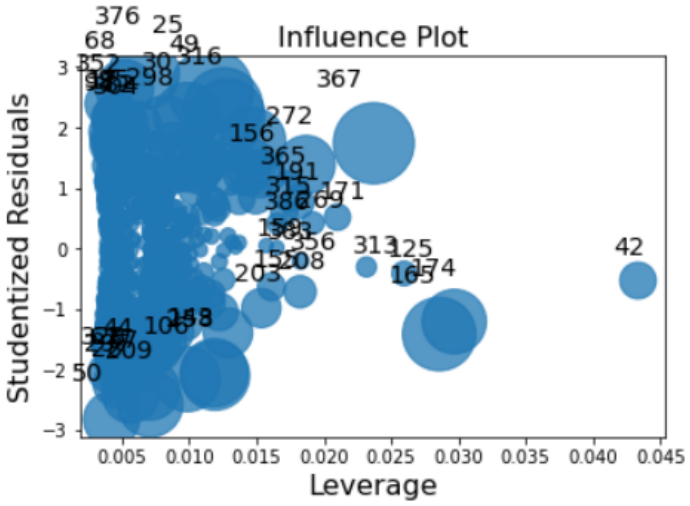
Şekil 21: Modelin katsayıları için %95 güven aralıkları

Sadeleştirilmiş modelde yer alan katsayılar için %95 güven aralıkları aşağıdaki gibidir:

Değişken	***Alt Sınır***	***Üst Sınır***
Intercept	*11.79*	*14.27*
US[Yes]	*0.69*	*1.71*
Price	*-0.065*	*-0.044*

*Tüm güven aralıkları sıfır değerini içermediğinden, bu katsayıların istatistiksel olarak anlamlı olduğu güvenle söylenebilir.Sadeleştirilmiş model için **influence plot** kullanılarak aykırı ve etkili gözlemler görselleştirilmiştir. Grafikte:*

- ***Cook's Distance** yüksek olan gözlemler, model tahminleri üzerinde orantısız etkiye sahip olabilir.*
- ***Leverage (kaldıraç)** değeri yüksek olan gözlemler, veri setinin uç noktalarında yer alabilir.*



Şekil 22: İnfluence Plot