2024-2025 AKADEMİK YILI

BAHAR DÖNEMİ

VERİ MADENCİLİĞİ

Ders Sorumlusu:

Dr. Günay TEMÜR

PIDD VERİ SETİ ÜZERİNDE KNN VE RANDOM FOREST MODELLERİNİN KARŞILAŞTIRMALI ÇALIŞMASI

Hazırlayanlar:

Şeyma ÇELİK

Bora ŞENEL

Özet:

Bu çalışmada veri madenciliği uygulamalarında sıklıkla kullanılan ve bir benchmarking veri seti olan Pima Indian Database veri seti üzerinde K-NN ve Random Forest modelleri kullanılarak diyabet hastalığının tahminlenmesine yönelik yayınlanmış bir makale baz alınarak, ilgili makalede kullanılmış olan modellemelerin ne ölçüde geliştirilebileceği ve eksikliklerinin neler olabileceği araştırılmak istenmiştir.

Bu çalışma bir karşılaştırma çalışması olduğundan, modellerin performansını ifade etmeye yönelik kullanılan metrikler, baz alınan makalede kullanılan metrikler ile özdeştir. Kullanılan performans metrikleri doğruluk (accuracy), kesinlik (precision), duyarlılık (recall) ve f1 skorudur.

1. Giriş:

Diyabet hastalığının tahminlenmesine yönelik çalışmalar özellikle veri madenciliği alanında kullanılan metotların karşılaştırılması noktasında literatürde oldukça geniş bir yayılıma ve hacme sahip çalışmalardandır. Genel bağlamda bu çalışmalar, veri madenciliği ve makine öğrenmesi yöntemlerinin hangisinin diyabet tahmini konusunda gerçeğe daha yakın sonuçlar verdiğini saptamaya çalışmak amacındadır. Hem bu çalışmada hem de baz alınan makalede kullanılan Pima Indian Diabetes veri setinin diyabet tahmini için kullanıldığı dar kapsamlı bir literatür taraması aşağıda yer almaktadır:

Yazarlar	Makalenin Adı	Kullanılan Yöntemler	Metrikler
(Gurunathan vd., 2023)	Web Application-based Diabetes Prediction using Machine Learning (Makine Öğrenmesi Kullanılarak Web Uygulama Tabanlı Diyabet Tahmini)	random forest, K-NN	accuracy, precision, recall ve f1 skoru
(Yamuna vd., 2022)	Diabetes Disease Prediction By Using Machine Learning Algorithms (Makine Öğrenme Algoritmalarını Kullanarak Diyabet Hastalığı Tahmini)	SVM, K-NN, DT(Karar Ağacı), LR (Lojistik Regresyon), Random Forest	accuracy, precision,recall
(Saxena vd., 2022)	A Novel Approach for Feature Selection and Classification of Diabetes Mellitus: Machine Learning Methods (Diyabet Mellitus'un Özellik Seçimi ve Sınıflandırılması İçin Yeni Bir Yaklaşım: Makine Öğrenmesi Yöntemleri)	multilayer perceptron (Çok katmanlı algılayıcı), DT (Karar Ağacı), Random Forest, K-NN	Recall,specificity (özgüllük), AUC, accuracy
(Tominanto vd., 2023)	Performance Analysis of Machine Learning Algorithms for Diabetes Diagnosis Prediction (Diyabet Tanı Tahmini için Makine Öğrenme Algoritmalarının Performans Analizi)	LR (Lojistik Regresyon), K-NN, Navie Bayes, DT, SVM, Random Forest, Gradient Boosting	ROC, accuracy, F-Measure, recall, precision

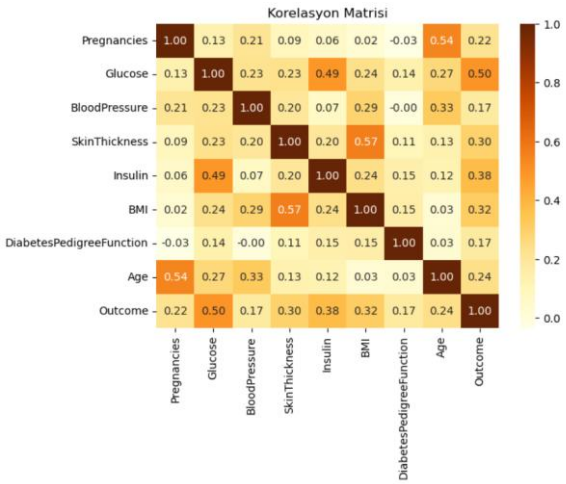
2. Metodoloji

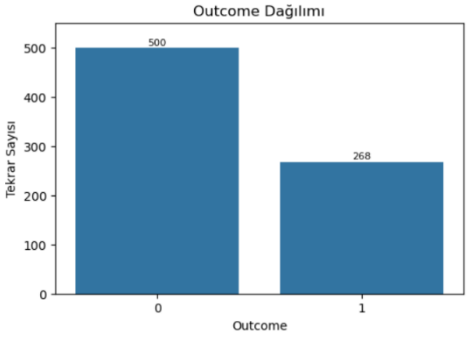
2.1 Veri Seti

Çalışmada veri kaynağı olarak PIDD (Pima Indian Diabetes Dataset) seti kullanılmıştır. Veri setine erişim Kaggle ve UCI gibi pek çok kanal aracılığıyla sağlanabilmektedir.

Kan basıncı (BloodPressure), cilt kalınlığı (SkinThickness), yaş (Age), hamilelikler (Pregnancies), insülin seviyesi (Insulin) , glikoz seviyesi (Glucose), diyabet soyağacı fonksiyonu (DiabetesPedigreeFunction) ve vücut kitle indeksi (BMI) ve çıktı (Outcome) nitelikleri veri setinin özelliklerini oluşturmaktadır.

Veri setine ait korelasyon matrisi ve çıktıların dağılımı, aşağıda yer aldığı üzeredir :





2.2 Veri Ön işleme

```
import pandas as pd
df = pd.read_csv('diabetes.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    float64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    float64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(4), int64(5)
memory usage: 54.1 KB
```

Verinin ön işlenmesi aşamasında, sol tarafta yer alan görselden da anlaşılabileceği üzere kullanılan veri setinin herhangi bir özelliğinde eksik değer/değerler bulunmadığından eksik değerlerin doldurulması işlemi gerçekleştirilmemiştir.

Ön işlemenin normalizasyon basamağında ise yalnızca KNN algoritması için bir normalizasyon işlemi gerçekleştirilmiştir. Bu işlem veri setinin RobustScaler ile ölçeklendirilmesini kapsamaktadır.

Random forest algoritması için veri seti üzerinde herhangi bir ölçeklendirme aracı uygulanımının -metrikler incelendiğinde - model performansını olumsuz etkilediği saptandığından normalizasyon aşaması es geçilmiştir.

2.3 Kullanılan Yöntemler :

2.3.1 KNN (K Nearest Neighbor)

KNN, temelde sınıflandırma işlemlerinde kullanımı tercih edilen ve aynı zamanda regresyon problemlerinde de faydalanılabilen, tembel öğrenir yapıda (lazy learning) bir makine öğrenmesi algoritmasıdır.

2.3.2 Random Forest

Bilinen karar ağaçlarından farklı olarak random forest algoritmasının arka planında oluşturulan her bir karar ağacı, eğitim verilerinin rastgele bir alt kümesiyle ve özelliklerin rastgele bir alt kümesiyle eğitilir. Random forest algoritması, oluşturulan ağaçların birbirinden bağımsız olmasına ve farklı kararlar alabilmesine olanak sağlaması açısından topluluk (ensemble) yaklaşımına sahip nitelikte bir makine öğrenmesi algoritmasıdır.

2.4 K-NN Modeli ve Uygulanması:

İlgili makalede yer alan K-NN modelinin özellikleri aşağıdaki gibidir:

- Veri seti %80 eğitim, %20 test olacak şekilde bölümlenmiş ve Min-Max Scaler ile ölçeklendirilmiştir.
- Mesafe ölçüm aracı olarak Öklid mesafesi tercih edilmiştir.
- Yapılan manuel hesaplamalar sonucunda optimal k değerinin 7 olduğu gözlemlenmiştir. (En yakın 4 komşu diyabetli, kalan 3'ü sağlıklı olacak şekilde.)
- Model performansı accuracy (doğruluk), precision (kesinlik), recall (duyarlılık) ve F-1 metrikleri ile değerlendirilmiş ve sonuç itibarıyla %73 accuracy, %87 precision, %73 recall ve %79 F-1 skoru eldesi sağlanmıştır.

Modele ait karışıklık matrisi :

	Actual Identified Diabetes	**Actual Unidentified Diabetes**
Predicted Classification Results Identified Diabetes	TP = 88	FP = 13
Predicted Classification Results Not Identified Diabetes	FN = 29	TN = 24

Uygulama 1 :

```
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

knn = KNeighborsClassifier(n_neighbors=7, metric='euclidean')
knn.fit(X_train_scaled, y_train)
```

İlk deneme, yalnızca ilgili makalede yer alan bilgiler ile bir model oluşturularak gerçekleştirilmiş ve bu modelin performansı aşağıdaki gibi ölçümlenmiştir.

```
Accuracy : 0.7792
Precision: 0.6923
Recall   : 0.6667
F1 Score : 0.6792
```

Performans metriklerinden de anlaşılacağı üzere ilgili makaledeki K-NN modelinin performansı, yalnızca makalede yer alan bilgileri dikkate alarak oluşturduğumuz K-NN modelimizden çok daha iyi performans göstermektedir. Bu durum, modelimiz üzerinde ek değişiklikler yapmamız gerektiği çıkarımında bulunmamıza sebep olmuştur.

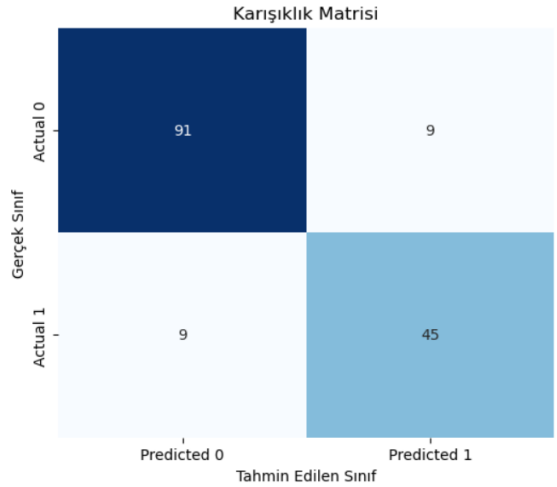
Uygulama 2 :

Bu model, oluşturduğumuz ilk modelin performans metriklerinin ilgili makaledeki metriklere göre iyileştirilmesi amacı güdülerek ve parametreler değiştirilerek oluşturulmuştur. Daha sağlıklı bir kıyaslama ve parametre değişimlerimizin etkisini gözlemleyebilmek adına eğitim ve test seti boyutunun dağılımı sabit tutulmuştur.

Parametre bazlı denemeler yapılmadan önce literatürde de sıkça başvurulan yöntemlerden biri ve hiper parametre optimizatörlerinden olan GridSearchCV kullanılarak ilk optimizasyon denemesi gerçekleştirilmiştir. Ancak GridSearchCV yalnızca optimal parametreleri bulma aracı olduğundan, verinin ölçeklendirilmesi noktasında doğrudan bir karar desteği sağlayamamaktadır. Ölçeklendirme aracı seçimi de model performansını fazlasıyla etkilediğinden, GridSearchCV optimizatörünün kullanımı sağ tarafta yer alan kod bloğunda görülebileceği üzeredir. Modele ait çıktı aşağıda yer almaktadır.

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('knn', KNeighborsClassifier())
])
param_grid = {
    'scaler': [StandardScaler(), MinMaxScaler(), RobustScaler()],
    'knn__n_neighbors': [3, 5, 7],
    'knn__metric': ['euclidean', 'manhattan'],
    'knn__weights': ['uniform', 'distance']
}
grid = GridSearchCV(pipe, param_grid, cv=5, scoring='f1', n_jobs=-1)
grid.fit(X_train, y_train)

print("En iyi parametreler:", grid.best_params_)

y_proba = grid.best_estimator_.predict_proba(X_test)[:, 1]
threshold = 0.5
y_pred_threshold = (y_proba >= threshold).astype(int)
```

```
En iyi parametreler: {'knn__metric': 'manhattan', 'knn__n_neighbors': 5, 'knn__weights': 'distance', 'scaler': RobustScaler()}
Test Accuracy: 0.8831
Test Precision: 0.8333
Test Recall: 0.8333
Test F1 Score: 0.8333
Karışıklık Matrisi
[[91  9]
 [ 9 45]]
```



Elde edilen çıktı sonucunda optimal k değeri 5, mesafe ölçüm aracı “manhattan”, ağırlıklandırma yöntemi olarak “distance”, ölçeklendirme aracı olarak ise “RobustScaler” olarak belirlenmiştir.

Ancak hiper parametre optimizatörleri, her ne kadar hızlı bir şekilde optimal sonucu verir nitelikteki parametreleri bulmaya çalışsalar da zaman zaman bu algoritmaların sunduğu parametrelerden daha iyi sonuç verici parametreler bulunabilmektedir –“Grid Search, belirli bir performans metriği üzerinden en iyi kombinasyonu seçer, ancak bu kombinasyonun genel en iyi çözüm olduğunu garantileyemez. Özellikle veri seti ve problem özelinde optimal bir çözüm bulma garantisi vermez. Başka bir deyişle, Grid Search tarafından seçilen kombinasyonlar, belirli bir metrik üzerinden en iyi performansı gösterse de genel model performansını etkileyen diğer faktörleri göz ardı edebilir.” (Bulut, 2024, para. 18) –.

Bu çalışma, bir model iyileştirme çalışması olduğundan bu ihtimal de göz önünde bulundurularak ek birtakım denemelerin yapılması gerekli görülmüştür.

1. Veri önişleme aşaması bir modelin kurgulanması sürecinin ilk adımı olduğundan birincil olarak bu aşamanın bir parçası olan ölçeklendirme aracı değiştirilerek denemeler yapılmıştır.

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scalers = {
    'MinMaxScaler': MinMaxScaler(),
    'StandardScaler': StandardScaler(),
    'RobustScaler': RobustScaler()
}
results = {}
for scaler_name, scaler in scalers.items():
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    knn = KNeighborsClassifier(n_neighbors=7, metric='euclidean')
    knn.fit(X_train_scaled, y_train)
    y_pred = knn.predict(X_test_scaled)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    results[scaler_name] = {
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1
    }
```

```
--- MinMaxScaler ---
Accuracy: 0.7857
Precision: 0.6897
Recall: 0.7273
F1 Score: 0.7080

--- StandardScaler ---
Accuracy: 0.7922
Precision: 0.7018
Recall: 0.7273
F1 Score: 0.7143

--- RobustScaler ---
Accuracy: 0.8571
Precision: 0.8000
Recall: 0.8000
F1 Score: 0.8000

--- En İyi Sonuçlar ---
Accuracy: RobustScaler ile 0.8571
Precision: RobustScaler ile 0.8000
Recall: RobustScaler ile 0.8000
F1 Score: RobustScaler ile 0.8000
```

Yukarıda ve solda yer almakta olan kod bloğu, mesafe ölçüm aracı ve k değeri sabit tutularak Min-Max Scaler, Robust Scaler ve Standard Scaler ölçeklendiricilerinin hangisinin modele daha iyi uyum sağladığını gözlemlemek amacıyla oluşturulmuştur. Modelin sağ tarafında ise performans metriklerine ait sonuçları görebilmektesiniz. Sonuçlardan da anlaşılacağı üzere Robust Scaler ölçeklendiricisi k değeri ve mesafe ölçüm aracı sabit tutulduğunda diğer ölçeklendirme araçlarından daha iyi sonuçlar vermektedir.

2. Bir sonraki aşamada mesafe ölçüm aracı seçimi üzerine denemeler yapılmıştır. İlk denemeler, bir önceki deneme dikkate alınarak Robust Scaler ve k değeri sabit tutularak gerçekleştirilmiştir.

```python
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

scaler = RobustScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
metrics = ['euclidean', 'manhattan', 'chebyshev', 'minkowski']
results = {}

for metric in metrics:
    if metric == 'minkowski':
        knn = KNeighborsClassifier(n_neighbors=7, metric=metric, p=3)
    else:
        knn = KNeighborsClassifier(n_neighbors=7, metric=metric)

    knn.fit(X_train_scaled, y_train)
    y_pred = knn.predict(X_test_scaled)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
```

Solunuzda yer alan kod bloğunda mesafe ölçüm aracı harici parametreler sabit tutularak “Euclidean”, “Manhattan”, “Chebyshev” ve “Minkowski” mesafe ölçümleyicileri karşılaştırılmıştır. Model sonucunda aşağıdaki çıktılar elde edilmiştir.

```
--- En İyi Sonuçlar ---
Accuracy: euclidean ile 0.8506
Precision: euclidean ile 0.8039
Recall: minkowski ile 0.7963
F1 Score: minkowski ile 0.7890
```

Gerçekleştirilen çalışma diyabet tahmini üzerine olduğundan her ne kadar accuracy ve precision değerleri öklidyen mesafe için sayısal anlamda daha iyi sonuçlar vermiş olsalar da yanlış negatifler, hastalığı olan bireylerin tespit edilememesine yol açacağından, sağlık alanında bu tür hataların minimize edilmesi kritik öneme sahiptir. Dolayısıyla recall değeri

dikkate alınarak model için en uyumlu tandemin bu aşamada Robust Scaler ve Minkowski mesafesi olacağı çıkarımında bulunulmuştur.

3. Bu aşamada Robust Scaler haricindeki ölçeklendirme araçlarının performansının öklidyen harici mesafe ölçüm araçları ile olan birlikteliğinin performansının göz ardı edilmemesi adına önceki iki aşamanın çaprazlaması sayılabilecek bir deneme daha yapılmıştır.

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

scalers = {
    'StandardScaler': StandardScaler(),
    'RobustScaler': RobustScaler(),
    'MinMaxScaler': MinMaxScaler()
}
metrics = {
    'euclidean': {},
    'manhattan': {},
    'chebyshev': {},
    'minkowski_p3': {'p': 3}
}

threshold = 0.5
results = {}
for scaler_name, scaler in scalers.items():
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    for metric_name, params in metrics.items():
        key = f"{scaler_name} + {metric_name}"
        print(f"--- Deneme: {key} ---")

        if metric_name == 'minkowski_p3':
            knn = KNeighborsClassifier(n_neighbors=7, metric='minkowski', weights='distance', p=params['p'])
        else:
            knn = KNeighborsClassifier(n_neighbors=7, metric=metric_name, weights='distance')

        knn.fit(X_train_scaled, y_train)
        y_probs = knn.predict_proba(X_test_scaled)[:, 1]

        y_pred = (y_probs >= threshold).astype(int)
```

Yukarıda yer alan kod bloğunda hem Standard, Robust ve Min-Max Scaler ölçeklendiricileri hem de “Euclidean”, “Manhattan”, “Chebyshev” ve “Minkowski” mesafe ölçümleyicileri iki boyutlu bir biçimde karşılaştırılmıştır. Elde edilen sonuçlar ise aşağıdaki gibidir.

```
--- En İyi Sonuçlar ---
Accuracy: StandardScaler + manhattan ile 0.8636
Precision: StandardScaler + manhattan ile 0.8113
Recall: StandardScaler + manhattan ile 0.7963
F1 Score: StandardScaler + manhattan ile 0.8037
```

Yukarıda yer alan sonuçlardan da anlaşılacağı üzere her ne kadar Robust Scaler ve Minkowski mesafesi ayrık olarak test edildiğinde daha iyi sonuçlar verseler de Standard Scaler ve Manhattan mesafesi tandemi modele daha iyi uyum sağlamaktadır.

4. Bu aşamada ise öncesinde yapılan test sonuçları dikkate alınarak, Standard Scaler ve manhattan mesafesi yöntemleri sabit tutularak k değeri üzerinde denemeler yapılmış olup aşağıda yer alan kod bloğu üzerinden 3 ile 10 arasındaki k değerlerinin performansı gözlemlenmiştir.

```python
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

threshold = 0.5
best_metrics = {
    'accuracy': {'score': 0, 'k': None},
    'precision': {'score': 0, 'k': None},
    'recall': {'score': 0, 'k': None},
    'f1': {'score': 0, 'k': None}
}
print(f"{'k':>3} | {'Accuracy':>8} | {'Precision':>9} | {'Recall':>7} | {'F1 Score':>8}")
print("-" * 45)

for k in range(3, 11):
    pipeline = Pipeline([
        ('scaler', StandardScaler()),
        ('knn', KNeighborsClassifier(n_neighbors=k, metric='manhattan', weights='distance'))
    ])
```

```
  k | Accuracy | Precision |  Recall | F1 Score
---------------------------------------------
  3 |   0.8636 |    0.8113 |  0.7963 |   0.8037
  4 |   0.8636 |    0.8235 |  0.7778 |   0.8000
  5 |   0.8701 |    0.8269 |  0.7963 |   0.8113
  6 |   0.8571 |    0.8333 |  0.7407 |   0.7843
  7 |   0.8636 |    0.8113 |  0.7963 |   0.8037
  8 |   0.8442 |    0.8000 |  0.7407 |   0.7692
  9 |   0.8506 |    0.7719 |  0.8148 |   0.7928
 10 |   0.8701 |    0.8400 |  0.7778 |   0.8077

En iyi k değerleri:
Accuracy : k = 5 (Score = 0.8701)
Precision: k = 10 (Score = 0.8400)
Recall   : k = 9 (Score = 0.8148)
F1       : k = 5 (Score = 0.8113)
```

Sol tarafta k değeri üzerinde yapılan denemelerin metriklere yansımalarını görmektesiniz. Accuracy ve F1 skoru metriklerini maksimize eden k değeri 5 olduğundan ve Precision ile Recall değerleri için eş k değerleri optimal olarak çıktılandırılmadığından, bu iki değerin orta noktası olarak değerlendirilebilecek F1 skoru metriği için de harici metrik olan Accuracy ile aynı k değerleri önerildiğinden model oluşturulurken k parametresi için 5 değeri tercih edilecektir.

5. Bir sonraki ve son optimizasyon çalışması mesafe ağırlıklandırma yöntemleri üzerinde olacaktır. Bu minvalde sklearn kütüphanesinin sunmuş olduğu “uniform” ve “distance” mesafelendiricilerinin F1 skoruna olan etkisini gözlemlemek amacıyla aşağıdaki kod bloğu kullanılmıştır :

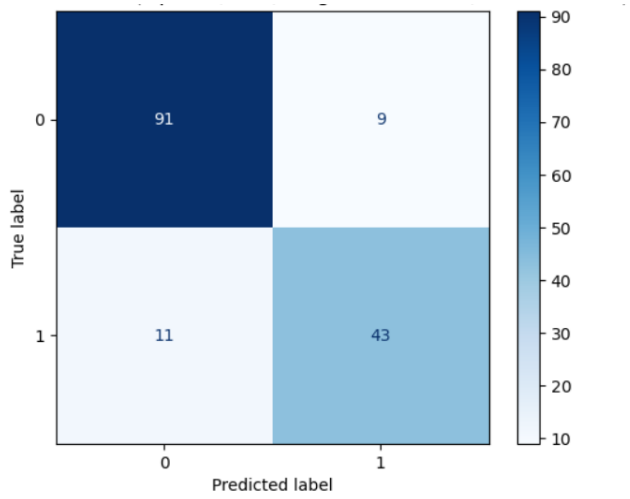
```python
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

threshold = 0.5
k = 5
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

weight_methods = ['uniform', 'distance']
best_f1 = 0
best_weight = None
best_metrics = {}
best_cm = None
best_y_pred = None

for w in weight_methods:
    knn = KNeighborsClassifier(n_neighbors=k, metric='manhattan', weights=w)
    knn.fit(X_train_scaled, y_train)

    y_probs = knn.predict_proba(X_test_scaled)[:, 1]
    y_pred = (y_probs >= threshold).astype(int)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, zero_division=0)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    print(f"Weight method: {w}")
    print(f"Accuracy : {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall   : {recall:.4f}")
    print(f"F1 Score : {f1:.4f}")
    print("-" * 30)

 if f1 > best_f1:
    best_f1 = f1
    best_weight = w
    best_metrics = {
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1': f1
    }
    best_cm = confusion_matrix(y_test, y_pred)
    best_y_pred = y_pred
```

Kod bloğunun çıktısı aşağıda yer almaktadır:

```
Weight method: uniform
Accuracy : 0.8636
Precision: 0.8235
Recall   : 0.7778
F1 Score : 0.8000
------------------------------
Weight method: distance
Accuracy : 0.8701
Precision: 0.8269
Recall   : 0.7963
F1 Score : 0.8113
```

Sonuçlardan da anlaşılabileceği üzere “distance ağırlıklandırmanın kullanımı “uniform” ağırlıklandırmadan sayısal anlamda daha yüksek niceliklerde metrik eldesi sağlanımına sebep olmaktadır. Dolayısı ile ağırlıklandırma için “distance” tercih edilecektir.

**Uygulama 2’nin son haline ait karışıklık matrisi aşağıdaki gibidir:*



Karar Aşaması :

Kullanılan Model	Accuracy	Precision	Recall	F1 Skoru
Uygulama 1 – baz alınan çalışma	0.7792	0.6923	0.6667	0.6792
Uygulama 2 – GridSearchCV	**0.8831**	**0.8333**	**0.8333**	**0.8333**
Uygulama 2 – manuel optimizasyon	0.8701	0.8269	0.7963	0.8113

GridSearchCV yönteminin önerdiği parametreler – k = 5, mesafe ölçüm aracı "manhattan", ağırlıklandırma yöntemi olarak "distance", ölçeklendirme aracı olarak "RobustScaler" - kullanılarak oluşturulan K-NN modeli tabloda yer alan diğer modellemelere kıyasla optimal sonucu vermektedir. Dolayısıyla nihai model GridSearchCV'nin önerdiği parametreler ile oluşturulmuştur.

2.5 Random Forest Modeli ve Uygulanması:
İgili makalede yer alan Random Forest modelinin özellikleri aşağıdaki gibidir:

- Veri seti %80 eğitim, %20 test olacak şekilde bölümlenmiş ve Min–Max Scaler ile ölçeklendirilmiştir.
- Model oluşturulurken random state kullanılmamış, bunun yerine 300 kez rastgele veri seçilmiştir.
- Modelde 100 adet karar ağacı (estimators) kullanılmıştır.
- Her düğümde rastgele seçilen 2 özellik ile ağaçlar oluşturulmuştur. (Örneğin Glukoz & Kan Basıncı, Deri Kalınlığı & VKİ)
- Model performansı accuracy (doğruluk), precision (kesinlik), recall (duyarlılık) ve F-1 metrikleri ile değerlendirilmiş ve sonuç itibarıyla %77 accuracy, %89 precision, %78 recall ve %83 F-1 skoru eldesi sağlanmıştır.

Makaledeki modele ait karışıklık matrisi :

	Actual Identified Diabetes	**Actual Unidentified Diabetes**
Predicted Classification Results Identified Diabetes	TP = 90	FP = 11
Predicted Classification Results Not Identified Diabetes	FN = 24	TN = 29

True Positives (TP = 90): 114 gerçek diyabetli örnekten 90'ını doğru olarak "diyabetli" diye sınıflandırmış.

False Negatives (FN = 24): 114 gerçek diyabetli örnekten 24'ünü "sağlıklı" olarak sınıflandırmış.

True Negatives (TN = 29): 40 gerçek sağlıklı örnekten 29'unu doğru "sağlıklı" olarak yakalamış.

False Positives (FP = 11): 40 sağlıklıdan 11'ini yanlışlıkla "diyabetli" olarak sınıflandırmış. Bu da yaklaşık olarak %72,5 demek.

Ana çıkarımlar

Model, diyabetli vakaları yakalamada (%78 recall) makul ama idealin biraz altında.

Yanlış alarm oranı (FP) düşük kalmış yani sağlıklı kişileri gereksiz yere tedaviye yönlendirme riski %27 civarı.

Genel doğruluk (accuracy) %77, kesinlik (precision) %89; yani "diyabet" dediğinde büyük çoğunlukla haklı.

Bu dağılım, özellikle yanlış negatifleri (kaçırılan hastaları) azaltacak ek adımlar —örneğin eşik ayarı veya dengesiz veri stratejileri— düşünülebileceğini gösteriyor.

Uygulama 1 :

```
n_runs = 300

acc_list  = []
prec_list = []
rec_list  = []
f1_list   = []

for seed in range(n_runs):
    X_train, X_test, y_train, y_test = train_test_split(
        X, y,
        test_size=0.2,
        random_state=seed
    )
    scaler = MinMaxScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled  = scaler.transform(X_test)

    rf = RandomForestClassifier(
        n_estimators=100,
        max_features=2,
        random_state=42
    )
    rf.fit(X_train_scaled, y_train)
```

İlk deneme, yalnızca ilgili makalede yer alan bilgiler ile bir model oluşturularak gerçekleştirilmiş ve bu modelin performansı aşağıdaki gibi ölçümlenmiştir.

```
Accuracy:  87.95% ± 2.51%
Precision: 83.22% ± 5.20%
Recall:    81.78% ± 5.38%
F1 Score:  82.34% ± 3.86%
```

Makaledeki parametreler baz alınarak oluşturulan ilk modelde; makaledeki performans metrikleriyle eş olmayan ve dahi daha yüksek accuracy (+%10), daha yüksek recall (+%3) ve F1 Skorunun neredeyse eşit durumda olduğu sonuçlar elde edilmiştir. Recall değeri artış göstermesine rağmen F1 Skorunun neredeyse aynı seviyede olmasının sebebi ise, precision skorumuzun (%5) düşüş göstermiş olmasıdır.

Metrik/Model	baz makaleye ait model	Uygulama 1 – baz parametreler
Accuray	%77	**%87,95**
Precision	**%89**	%83,22
Recall	%78	**%81,78**
F1 Skoru	**%83**	%82,34

**Diğer uygulamalar birbirleri ile karşılaştırılırken random state değeri 42 kabul edilmiştir.*

Uygulama 2

```python
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42
)

rf = RandomForestClassifier(
    n_estimators=100,
    max_features=2,
    random_state=42
)
rf.fit(X_train, y_train)

y_pred = rf.predict(X_test)
```

Bu seferki modelimizde sadece Min-Max Scaler'ı kaldırdık. Diğer değerleri örneğin estimator değerini değiştirmedik.

```
Accuracy  : 88.31%
Precision: 81.36%
Recall    : 87.27%
F1 Score  : 84.21%

Confusion Matrix:
[[88 11]
 [ 7 48]]
```

Accuracy değerimiz makaledeki değere göre %0.36 arttı, Precision değerimiz %1,86 azalırken Recall değerimiz %5,79 arttı. Bu sayede F1 Skorumuz %1,87 artmış oldu.

Metrik/Model	baz makaleye ait model	Uygulama 1 – baz parametreler	Uygulama 2 - (Min-Max Kaldırılmış)
Accuray	%77	%87,95	**%88,31**
Precision	**%89**	%83,22	%81,36
Recall	%78	%81,78	**%87,27**
F1 Skoru	%83	%82,34	**%84,21**

Uygulama 3 En İyi Sonuçları Veren Model

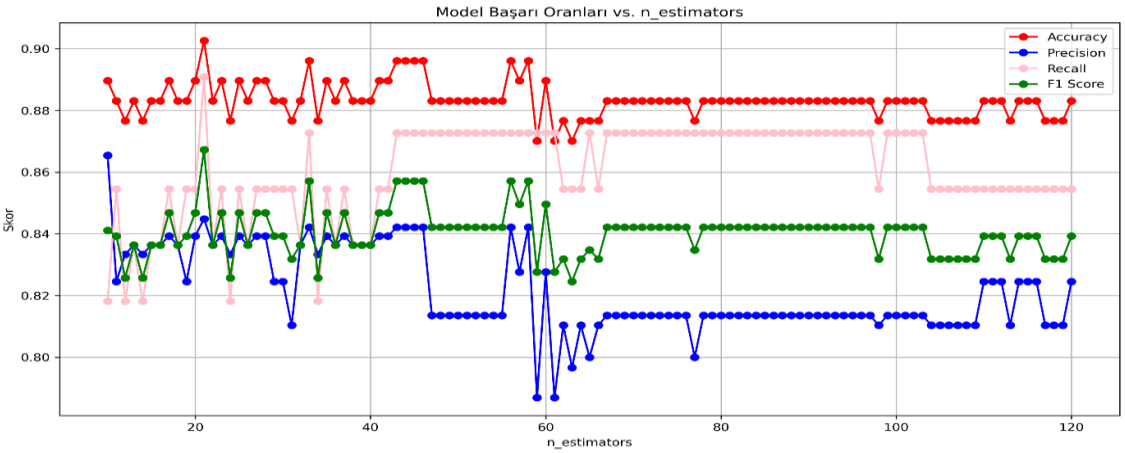
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42
)

rf = RandomForestClassifier(
    n_estimators=56,
    max_features=2,
    random_state=42
)
rf.fit(X_train, y_train)

y_pred = rf.predict(X_test)
```

```
for n in range(10, 121):
    rf = RandomForestClassifier(
        n_estimators=n,
        max_features=2,
        random_state=42
    )
```

Bu model kapsamında yapılan temel değişiklik, modelde kullanılan ağaç sayısı olan n_estimators parametresinin optimize edilmesidir. Literatürde baz alınan çalışmada bu değer 100 olarak sabitlenmişken, bu çalışmada 10 ile 121 arası değerler sistematik olarak test eden kod yazılarak modelin hangi n_estimators değeriyle en yüksek başarıya ulaştığı test edilmiştir.

Başarı Oranı

n_estimators	Accuracy	Precision	Recall	F1 Score
10	0.8896	0.8654	0.8182	0.8411
11	0.8831	0.8246	0.8545	0.8393
12	0.8766	0.8333	0.8182	0.8257
13	0.8831	0.8364	0.8364	0.8364
14	0.8766	0.8333	0.8182	0.8257
15	0.8831	0.8364	0.8364	0.8364
16	0.8831	0.8364	0.8364	0.8364
17	0.8896	0.8393	0.8545	0.8468
18	0.8831	0.8364	0.8364	0.8364
19	0.8831	0.8246	0.8545	0.8393
20	0.8896	0.8393	0.8545	0.8468
21	0.9026	0.8448	0.8909	0.8673
22	0.8831	0.8364	0.8364	0.8364
23	0.8896	0.8393	0.8545	0.8468
24	0.8766	0.8333	0.8182	0.8257



Yapılan denemeler sonucunda n_estimators=21 değeri ile en yüksek doğruluk, duyarlılık ve F1 skoru değerlerine ulaşılmış; bu da daha az sayıda ama daha etkili karar ağacıyla modelin daha isabetli tahminlerde bulunabildiğini göstermiştir. Bu hiper parametre seçimiyle modelde Accuracy değerimiz önceki modele göre %2,3, Precision değerimiz %3,12 Recall değerimiz %1,82 arttı. Precision ve Recall değerlerimizin yükselmesi sonucu F1 Skorumuz da %2,52 artmış oldu.

```
Accuracy : 90.26%
Precision: 84.48%
Recall   : 89.09%
F1 Score : 86.73%

Confusion Matrix:
[[90  9]
 [ 6 49]]
```

Metrik/Model	baz makaleye ait model	Uygulama 1- baz parametreler	Uygulama 2 - min-max kaldırılmış	Uygulama 3- estimator = 21
Accuracy	%77	%87,95	%88,31	**%90,61**
Precision	**%89**	%83,22	%81,36	%84,48
Recall	%78	%81,78	**%87,27**	**%89,09**
F1 Skoru	%83	%82,34	%84,21	**%86,73**

SONUÇ

Oluşturmuş olduğumuz KNN ve Random Forest modellerimiz istatistikî açıdan baz alınan makaleye kıyasla genel anlamda daha yüksek başarı oranlarına sahiptirler. Metrikler üzerindeki değişimi daha net gözlemleyebilmek adına literatür taramamızda yer verdiğimiz – aynı veri seti üzerindeki diğer çalışmalar – da dahil edilerek aşağıda yer alan tablolar oluşturulmuştur:

KNN algoritması için;

Metrik/Model	Baz makale	Bize ait	(Gurunathan vd., 2023)	Yamuna vd., 2022)	Saxena vd., 2022)	Tominanto vd., 2023
accuracy	0.7792	**0.8831**	0.7533	0.69	0.7858	0.8051
precision	0.6923	**0.8333**	0.7636	0.60	-	0.78
recall	0.6667	**0.8333**	0.7826	0.41	0.7856	0.64
F1 skoru	0.6792	**0.8333**	0.7730	-	-	0.70

Yorum:

Bu çalışma kapsamında oluşturulmuş olan KNN modeli, baz alınan makale de dahil olmak üzere tabloda yer alan kendisi haricindeki tüm çalışmalardan daha başarılı bir sınıflandırma işlemi gerçekleştirmektedir. Oluşturulan model için baz alınan makalede kullanılan model üzerinden oransal bir karşılaştırma yapılmak istenirse oluşturulan modelin; accuracy metriği için %10,39 ; precision metriği için %14,1 ; recall metriği için %16,6 ve son olarak f1 skoru için ise %15,41 daha başarılı performans gösterdiği söylenilebilmektedir.

Random Forest algoritması için;

Metrik/Model	Baz makale	Bize Ait	(Gurunathan vd., 2023)	Yamuna vd., 2022)	Saxena vd., 2022)	Tominanto vd., 2023
accuracy	%77	**%90,61**	0.8181	%80	%79,83	0.7922
precision	**%89**	%84,48	0.7209	%74	-	0.69
recall	%78	**%89,09**	0.6595	%50	%79,8	0.75
F1 skoru	%83	**%86,73**	0.6888	-	-	0.72

Yorum:

Random Forest algoritması için gerçekleştirdiğimiz uygulamalarda, modelin hiper parametreleri üzerinde yapılan değişikliklerin doğrudan model başarısına etki ettiği gözlemlenmiştir. Özellikle makalede kullanılan Min-Max Scaler'ın modelden çıkarılması ve n_estimators parametresinin optimize edilmesiyle modelin doğruluk, duyarlılık (recall) ve F1 skoru metriklerinde anlamlı artışlar elde edilmiştir. En iyi sonuçları veren modelde %90.61 doğruluk, %84.48 precision, %89.09 recall ve %86.73 F1 skoru elde edilmiş; bu değerler baz alınan literatürdeki modele kıyasla tüm metriklerde daha yüksek performans göstermiştir.

Modelin yüksek recall değerine ulaşması, diyabetli bireylerin doğru şekilde sınıflandırılması açısından son derece kritik bir başarıyı işaret etmektedir.

Oluşturmuş olduğumuz modellerinin performans metrikleri göz önünde bulundurulduğunda, oluşturduğumuz Random Forest algoritmasının KNN algoritmasından daha başarılı bir şekilde tahminleme işlemi gerçekleştirdiği gözlemlenmiştir.

Kaynakça

1. Bulut, F. (2024). Model optimizasyonunda yol haritası: Grid Search yöntemi ve gelecek perspektifler. *MCBÜ Veri Topluluğu*. Erişim tarihi: 25 Nisan 2025,
2. https://veri.mcbu.edu.tr/article-detail/10/
3. P. T. Siva Gurunathan, R. S, R. S and N. S, "Web Application-based Diabetes Prediction using Machine Learning," *2023 7th International Conference on Computing Methodologies and Communication (ICCMC)*, Erode, India, 2023, pp. 296-302.
4. V.Yamuna ,D.Ushanthi,,B.Krishna chaitanya ,Y.Divya sri ,T.Jagadish, Diabetes Disease Prediction By Using Machine Learning Algorithms, *International Journal Of Creative Research Thoughts*, 2022, Volume 10, Issue 6.
5. Saxena, Roshi, Sharma, Sanjay Kumar, Gupta, Manali, Sampada, G. C., A Novel Approach for Feature Selection and Classification of Diabetes Mellitus: Machine Learning Methods, *Computational Intelligence and Neuroscience*, 2022, 3820360, 11 pages, 2022.
6. T. Tominanto, A. Syukur, Affandy and A. Marjuni, "Performance Analysis of Machine Learning Algorithms for Diabetes Diagnosis Prediction," *2023 International Seminar on Application for Technology of Information and Communication (iSemantic)*, Semarang, Indonesia, 2023, pp. 227-232, doi: 10.1109/iSemantic59612.2023.10295368.