Makine Öğrenimi Algoritmalarını Kullanarak Diyabet Tahmini

Yazılım Mühendisliği (Uzaktan Eğitim) Ana Bilim Dalı

Yüksek Lisans Projesi

Kürşat Osman Alımlı

ORCID 0000-0000-0000-0000

Proje Danışmanı: Doç. Dr. Aytuğ ONAN

Aralık 2023

Makine Öğrenmesi Algoritmalarını Kullanarak Diyabet Tahmini

Öz

Makine Öğrenmesi Algoritmalarını Kullanarak Diabet Tahmini yapabilmek üzere belirlenmiş olan Makine Öğrenimi Algoritmalarını kullanarak diabet tahminleme modelleri üzerinde çalışılmıştır.

Anahtar Sözcükler: Makine öğrenmesi, teknoloji, yapay zeka, tıp, diabet

Teşekkür

Proje çalışmasının ortaya çıkmasında emekleri olan Doç. Dr. Aytuğ ONAN'a teşekkür ederim.

İçindekiler

Şekiller Listesi

Tablolar Listesi

Kısaltmalar Listesi

KNN	K-Nearest Neighbors
NB	Naive Bayes
SVM	Support Vector Machine
DT	Decission Trees
RFC	Random Forest Classifier

Giriş

1 Projenin Amacı

> "*Diyabet (DM), insülin üretme ve insüline tepki verme yeteneğinin engellendiği, kan dolaşımında yüksek glukoz seviyelerine neden olabilen en yaygın hastalıklardan biridir (Lonappan et al., 2007). Diyabetle birlikte Koroner Arter Hastalığı (CAD), Koroner Böbrek Hastalığı (CKD), Kronik obstrüktif solunum hastalığı (COPD), Hipertansiyon (HTN) ve Hipotiroidizm gibi çeşitli hastalıklar ortaya çıkabilir. Bu hastalıklar belli bir aşamaya gelene kadarbelirgin belirtiler göstermez. Bu nedenle, bu hastalıkların erken teşhisi ve uygun tedavisi hastanın daha iyi bir duruma kavuşmasına yardımcı olabilir.* (Kang et al., 2013)"

> "*Diyabet iki sınıfa ayrılabilir: Tip 1 diyabet (T1D) ve Tip 2 diyabet (T2D). Tip 1 diyabetli hastalar genellikle daha genç yaşta, genellikle 30 yaşın altındadır. Tipik klinik belirtiler artmış susuzluk ve sık idrara çıkma, yüksek kan glukoz seviyeleridir (Iancu et al., 2008). Bu tür diyabet sadece oral ilaçlarla etkili bir şekilde tedavi edilemez ve hastaların insülin tedavisine ihtiyaçları vardır. Tip 2 diyabet genellikle orta yaşlı ve yaşlı insanlarda daha sık görülür ve sıklıkla obezite, hipertansiyon, dislipidemi, arteriyoskleroz ve diğer hastalıklarla ilişkilidir.* (Robertson et al., 2011)"

Dünya Sağlık Örgütü'ne (WHO) göre, düşük gelirli veya pasif gelirli ülkelerde yaklaşık 422 milyon kişi diyabet hastasıdır. 2030 yılına kadar bu sayının 490 milyona çıkması beklenmektedir. Bununla birlikte, diyabet Çin, Kanada, Hindistan gibi farklı ülkelerde yaygındır.

Bugün diyabet, dünyada ölümün önemli bir nedenidir. Diyabet gibi hastalıkların erken teşhisi hayat kurtarabilir. Günümüzde, makine öğrenimi teknikleri kanser, diyabet, kalp hastalığı, tiroid gibi çeşitli hayati tehlike oluşturan hastalıkların tahmin edilmesi veya teşhisi için kullanılmaktadır.

Bu çalışmanın temel amacı, en yüksek doğrulukla en iyi sınıflandırıcıyı bulmak için çeşitli makine öğrenimi tekniklerini kullanarak diyabet tahminini ve bu tekniklerin çıktı analizini yapmaktır. Bu çalışma, diyabet hastalığıyla ilgili farklı özellikleri baz alarak diyabet tahminini incelemektir.

2 Veri Seti

2.1 Veri Seti Hakkında

Bu çalışmada Pima Indian Diabetes Veri Seti kullanılmıştır. Bu veri seti, California Üniversitesi, Irvine UCI AI veri deposundan erişilebilir ve kullanılabilir durumdadır.

Kullanılan veri setindeki örnekler daha büyük bir veritabanından çeşitli kısıtlamalar seçilerek hazırlanmıştır. Veri setinde yer alan tüm hastalar en az 21 yaşında Pima Kızılderili genlerine sahip kadınlardır.

Bu veri setinde, sonuç özniteliği de dahil olmak üzere dokuz öznitelik bulunan 768 kayıt bulunmaktadır. Sonuç olarak, bu kayıtların 268'i "pozitif sonuç" olarak belirtilmiş olup hastanın diyabete sahip olduğunu gösterirken, 500 kayıt "negatif sonuç" olarak belirtilmiş, yani hastaların diyabeti olmadığını göstermektedir.

2.2 Veri Seti Özellik Açıklamaları

Pregnancies: Gebelik Sayısı

Glucose(Glukoz): Ağızdan alınan bir glukoz tolerans testinin 2 saat sonrasındaki plazma glukoz konsantrasyonu.

Bu bir laboratuvar testidir ve vücudunuzun şekeri nasıl işlediğini kontrol etmek için yapılır. Normal bir kişinin (glukoz testinden 2 saat sonra) 140 mg/dl'den az olması beklenir.

Blood Pressure(Kan Basıncı): Diyastolik kan basıncı (mm Hg).

Normal değerler 80'den düşük olmalıdır.

Evre 1 hipertansiyon: 80-89

Evre 2 hipertansiyon: 90 veya daha fazla

Hipertansif Kriz: 120 veya daha fazla

Skinn Thickness(Cilt Kalınlığı): Triseps cilt kıvrım kalınlığı (mm).

Yetişkinler için normal değerler erkeklerde 2.5 mm, kadınlarda 18 mm'dir.

Insulin: 2 saatlik serum insülini (mu U/ml). İnsülin, kan şekeri taşımaya yardımcı olan bir hormondur.

150 mu U/ml kritik bir sayıdır ve tip 1 veya 2 diyabeti olan çoğu kişi insülin tedavisine ihtiyaç duyar.

BMI(Vücut Kitle Endeksi): Vücut ağırlığı (kg) / (boy uzunluğu (m) ^ 2) olarak hesaplanır ve bir kişinin kilolu mu yoksa zayıf mı olduğunu değerlendirir.

Zayıf: 18.5'ten düşük

Normal Kilolu: 18.5 - 24.9

Kilolu: 25-29.9

Obez: 30.0'dan fazla

Diabetes Pedigree Function(Diabet Soyağacı Fonksiyonu):

Akrabalardaki geçmiş hakkında bilgi sağlar. Bu, genetik etkiyi ölçen bir parametredir.

Outcome(Hedef Değişken): diyabeti gösterir; 0 diyabeti olmadığını gösterir.

3 Proje

Uygulamada IDE olarak Jupyter Notebook ve dil olarak Phtyon(3.10.9) kullanılmıştır.

3.1 Kütüphanelerin Uygulamaya Dahil Edilmesi

```
In [1]: import pandas as pd
        import numpy as np
        import sklearn as sk
        import seaborn as sns
        from IPython.display import display
        from matplotlib import pyplot as plt
        from sklearn import preprocessing
        from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score, roc_curve,precision_score,recall_score,classificatic
        from sklearn.model_selection import train_test_split
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.linear_model import LogisticRegression
        from sklearn.model_selection import StratifiedKFold
        from sklearn.model_selection import cross_val_score
        from sklearn.preprocessing import StandardScaler
        from sklearn.naive_bayes import GaussianNB
        from sklearn.svm import SVC
        from sklearn.pipeline import Pipeline
        from warnings import simplefilter
        simplefilter(action='ignore', category=FutureWarning)
```

Şekil 3.1.1: Kütüphanelerin tanımlanması

3.2 Veri Setinin Uygulamaya Aktarılması

```
In [25]: originalData = pd.read_csv('E:\Python Projeler\VeriBilimiOdev\diabetes.csv')
         originalData.head()
```

Out[25]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Şekil 3.2.1: Veri setinin projeye alınması

3.3 Veri Seti Ön İşlemleri

> "*En kritik işlem veri ön işlemedir. Bu işlem, veri kümesi üzerinde makine öğrenmesi yöntemlerini etkili bir şekilde uygulamak için güvenilir sonuçlar ve verimli tahminler için önemlidir*. (Soni et al., 2020)"

3.3.1 Null Veri Kontrolü

Veri setindeki null veriler median algoritması ile replace edilmiştir.

```
In [4]: #Replace null value by median
        def ReplaceNullValues(model):
            copyData = model.copy()
            cols = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI','Pregnancies','DiabetesPedigreeFunction','Age']
            copyData[cols] = model[cols].replace({0 : np.nan})
            copyData = model.fillna(model.median())
            copyData.describe()
            return copyData;
```

Şekil 3.3.1.1: Null veri kontrolü ve replacement

3.3.2 Normalizasyon

Verilerin normalizasyonunda MinMaxScaler algoritması kullanılarak 0-1 normalizasyonu yapılmıştır.

```
In [5]: # 0-1 Normalisation
        def NormalizeData(model):
            x = model.values
            min_max_scaler = preprocessing.MinMaxScaler()
            x_scaled = min_max_scaler.fit_transform(x)
            dataSet = pd.DataFrame(x_scaled)
            dataSet.columns=['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age','Ou
            dataSet.head()
            return dataSet;
```

Şekil 3.3.2.1: MinMaxScaler ile 0-1 normalizasyonu

3.3.3 Ön İşlemleri Tamamlanmış Veri Setinin Kontrolü

```
In [22]: normalizedData.sample(5)
```

Out[22]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
216	0.294118	0.547739	0.508197	0.414141	0.152482	0.533532	0.186166	0.066667	1.0
746	0.058824	0.738693	0.770492	0.414141	0.000000	0.734724	0.119556	0.100000	1.0
738	0.117647	0.497487	0.491803	0.171717	0.189125	0.545455	0.160120	0.000000	0.0
587	0.352941	0.517588	0.540984	0.000000	0.000000	0.362146	0.073015	0.133333	0.0
252	0.117647	0.452261	0.655738	0.141414	0.065012	0.363636	0.073015	0.050000	0.0

Şekil 3.3.3.1: Ön işlemleri tamamlanmış veri seti örnek gösterimi

3.4 Tahmin Modelleri

Veri kümemiz hazır olduğunda, Makine Öğrenmesi yöntemlerini kullanarak veri kümesini sınıflandırmak için kullanıyoruz. Bu makalede, Gebelik, Glukoz, Kan Basıncı, Deri Kalınlığı, İnsülin, BMI, Soykütüğü ve Yaş gibi özelliklerle KNN, DT, NB, SVM ve RFC sınıflandırma algoritmaları kullanılmıştır.

3.4.1 Eğitim ve Test Verilerinin Oluşturulması

Veri setinin 1/3'ü eğitim verisi olarak kullanılmıştır.

```
In [8]: def CreateTrainAndTestSets(model):

            sc_X = StandardScaler()
            X =  pd.DataFrame(sc_X.fit_transform(model.drop(["Outcome"],axis = 1),),
                columns=['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
               'BMI', 'DiabetesPedigreeFunction', 'Age'])
            y = model.Outcome
            return train_test_split(X,y,test_size=1/3,random_state=42, stratify=y)
```

Şekil 3.4.1.1: Eğitim ve test verilerinin hazırlanması

3.4.2 Sınıflandırma Raporları, Karmaşıklık Matrisleri ve Score

Sınıflandırma raporları, karmaşıklık matrisleri ve score için parametre olarak test verisi, tahmin ve sınıflandırıcı ismini parametre olarak alan üç fonksiyon tanımlanmıştır. İlgili fonksiyonlar aşağıda belirtilmiştir.

```
In [15]: def ShowScores(prediction,ytest,classifierName):
             accuracy = accuracy_score(ytest, prediction)
             precision = precision_score(ytest, prediction)
             recall = recall_score(ytest, prediction)
             fScore = f1_score(ytest,prediction)
             print(classifierName+" Accuracy:", accuracy)
             print(classifierName+" Precision:", precision)
             print(classifierName+" Recall:", recall)
             print(classifierName+" F1 Score:",fScore)
             print('\n')
```

Şekil 3.4.2.1: Score değerlerinin hesaplanması ve gösterimi

```
In [16]: def ShowClassificaionReport(ytest,ypred,classifierName):
             print(classifierName+'\n')
             print(classification_report(ytest, ypred))
```

Şekil 3.4.2.2: Sınıflandırma raporunun hazırlanması ve gösterimi

```
In [17]: def ShowConfussionMatrix(ytest, prediction,classifierName):



             conf_matrix = confusion_matrix(y_true=ytest, y_pred=prediction)
             fig, ax = plt.subplots(figsize=(7.5, 7.5))
             ax.matshow(conf_matrix, cmap=plt.cm.Blues)
             for i in range(conf_matrix.shape[0]):
                 for j in range(conf_matrix.shape[1]):
                     ax.text(x=j, y=i,s=conf_matrix[i, j], va='center', ha='center', size='xx-large')
             plt.xlabel('Predictions', fontsize=18)
             plt.ylabel('Actuals', fontsize=18)
             plt.title(classifierName + ' Confusion Matrix', fontsize=18)
             plt.show()
```

Şekil 3.4.2.3: Karmaşıklık matrisi oluşturma ve gösterimi

3.4.3 K-Nearest Neighbors

KNN (K-Nearest Neighbors), denetimli makine öğrenmesi alanında kullanılan bir sınıflandırma algoritmasıdır. Temel prensibi, veri noktalarını birbirine olan benzerliklerine göre gruplandırmaktır. KNN algoritmasında, sınıflandırılacak yeni bir veri noktası, en yakınında bulunan K sayıda komşusuyla karşılaştırılır. Bu komşuların sınıfları incelenir ve çoğunluğun sınıfı, yeni veri noktasının tahmin edilen sınıfı olarak atanır.

KNN algoritması, bir örneklem veri seti üzerinde eğitim yapar ve yeni verileri sınıflandırmak için bu eğitim setini kullanır. KNN'nin temel varsayımı, benzer özelliklere sahip veri noktalarının aynı sınıfa ait olma eğiliminde olduğudur. Bu nedenle, KNN, veriye dayalı bir karar verme süreci olarak kabul edilir.

KNN'nin önemli bir parametresi K değeridir. K, yeni bir veri noktasının kaç komşusu ile karşılaştırılacağını belirler. K değeri arttıkça sınıflandırma daha genelleşir, K değeri azaldıkça sınıflandırma daha spesifik hale gelir. KNN algoritması genellikle sınıflandırma problemlerinde kullanılır, ancak regresyon problemleri için de uyarlanabilir.

KNN'nin avantajları arasında basitliği, kolay uygulanabilirliği ve yüksek esneklik yer alır. Bununla birlikte, büyük veri setlerinde ve yüksek boyutlu verilerde hesaplama maliyeti yüksek olabilir ve veri kümesinin dengesiz dağılımı durumunda yanlılık oluşabilir.

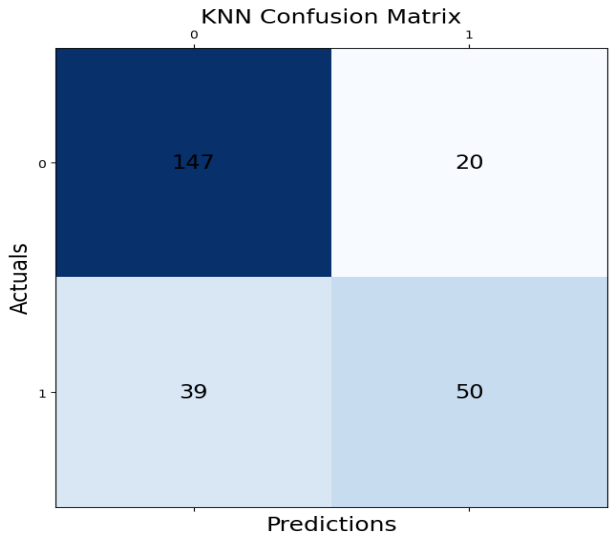
Bu makalede, Pima Indian Diabetes Veri Seti KNN algoritmasının yukarıda bahsedilen özellikleriyle test edilmiştir.

```
In [9]: def StartKNN(xtrain,ytrain,xtest,ytest):
            test_scores = []
            train_scores = []
            k_values = [i for i in range (1,15)]
            for i in k_values:

                knn = KNeighborsClassifier(i)
                knn.fit(xtrain,ytrain)

                train_scores.append(knn.score(xtrain,ytrain))
                test_scores.append(knn.score(xtest,ytest))
                max_train_score = max(train_scores)
            train_scores_ind = [i for i, v in enumerate(train_scores) if v == max_train_score]
            print('KNN: Max train score {} % and k = {}'.format(max_train_score*100,list(map(lambda x: x+1, train_scores_ind))))
            max_test_score = max(test_scores)
            test_scores_ind = [i for i, v in enumerate(test_scores) if v == max_test_score]
            print('KNN: Max test score {} % and k = {}'.format(max_test_score*100,list(map(lambda x: x+1, test_scores_ind))))

            sns.lineplot(x = k_values, y = test_scores, marker = 'o')
            plt.xlabel("K Values")
            plt.ylabel("Accuracy Score")

            knn = KNeighborsClassifier(n_neighbors=11)
            knn.fit(xtrain, ytrain)


            return  knn.predict(xtest)
```

Şekil 3.4.3.1: KNN sınıflandırma modelinin kullanımı

```
knn_Predict = StartKNN(X_train,y_train,X_test,y_test)
ShowScores(knn_Predict,y_test,'KNN')
ShowClassificaionReport(y_test,knn_Predict,'KNN')
ShowConfussionMatrix(y_test,knn_Predict,'KNN')
```

Şekil 3.4.3.2: KNN modeli sonuç fonksiyonlarının çalıştırılması



Şekil 3.4.3.3: KNN karmaşıklık matrisi

```
KNN: Max train score 100.0 % and k = [1]
KNN: Max test score 76.953125 % and k = [11]
KNN Accuracy: 0.76953125
KNN Precision: 0.7142857142857143
KNN Recall: 0.5617977528089888
KNN F1 Score: 0.6289308176100629


KNN

              precision    recall  f1-score   support

         0.0       0.79      0.88      0.83       167
         1.0       0.71      0.56      0.63        89

    accuracy                           0.77       256
   macro avg       0.75      0.72      0.73       256
weighted avg       0.76      0.77      0.76       256
```

Şekil 3.4.3.4: KNN modeli ile yapılan sınıflandırma sonuçları

3.4.4 Naive Bayes

Naive Bayes, makine öğrenmesi ve istatistiksel sınıflandırma problemlerinde yaygın olarak kullanılan bir sınıflandırma algoritmasıdır. Bu algoritma, Bayes teoremini temel alır ve basit bir varsayım olan "naive" (saf) varsayımı yapar.

Naive Bayes algoritması, giriş özelliklerinin birbirinden bağımsız olduğunu varsayar. Yani, her bir özelliğin sınıf etiketini belirlemedeki etkisi diğer özelliklerden bağımsız olarak değerlendirilir. Bu varsayım, özellikler arasındaki ilişkileri veya etkileşimleri göz ardı eder, bu nedenle "naive" olarak adlandırılır.

Algoritmanın çalışma prensibi şu şekildedir: İlk olarak, eğitim veri setindeki özelliklerin ve sınıf etiketlerinin istatistiksel dağılımları hesaplanır. Bu dağılımlar, sınıfların olasılık tahminlerini sağlar. Ardından, yeni bir veri noktasını sınıflandırmak için Bayes teoremini kullanır.

Naive Bayes algoritması, veri setlerinin büyüklüğüne bağımlı olarak hızlı ve verimli bir şekilde çalışır. Ayrıca, sınıflandırma problemlerinde iyi bir performans gösterebilir, özellikle metin sınıflandırma gibi metin tabanlı problemlerde yaygın olarak kullanılır. Naive Bayes, düşük boyutlu veri setlerinde ve hızlı bir tahmin yapılması gereken uygulamalarda tercih edilen bir algoritmadır.

Ancak, Naive Bayes algoritmasının "naive" varsayımı nedeniyle, özellikler arasındaki gerçek bağımlılıkları veya etkileşimleri dikkate almadığından bazı durumlarda performansı düşük olabilir. Özellikle, özellikler arasında güçlü bir bağımlılık veya etkileşim olduğu karmaşık veri setlerinde doğruluk oranı düşebilir. Ayrıca, eğitim veri setinde nadir veya sıfır olasılıklı özellik değerleri varsa, algoritma bu özellikleri doğru bir şekilde modelleyemez.

Naive Bayes Denklemi

$P(c|x) = P(x|c)P(c) / P(x)$

$P(c\ |X) = P(x1|c)\square\ P(x2|c\ )$x . . . x$P(x2\square\ c)\square P(c)$

Nerede:

P(c|X) sonsal Olasılıktır.

P(x|c) Olabilirliktir.

P(c) Sınıf Öncül Olasılığıdır.

P(x) Tahmin Edici Öncül Olasılığıdır.

Naive Bayes Sözde Kodu

Eğitim veri kümesi T,

F= (f1, f2, f3,..., fn) // test veri kümesindeki tahmin edici değişkenin değeri.

Çıktı:

Test veri kümesinin bir sınıfı.

Adım:

1. Eğitim Veri Kümesi T'yi okuyun;

2. Her sınıfın tahmin edici değişkenlerinin ortalamasını ve normunu hesaplayın;

3. Tekrarla

4. Her sınıfta gauss yoğunluğu denklemini kullanma olasılığının hesaplanması;

5. Tüm tahmin edici değişkenlerin (f1, f2, f3,..., fn) olasılığının tahminini bekleyene kadar,.

6. İlgili sınıf için olabilirlik hesaplanır;

7. En yüksek olasılığı elde edin;

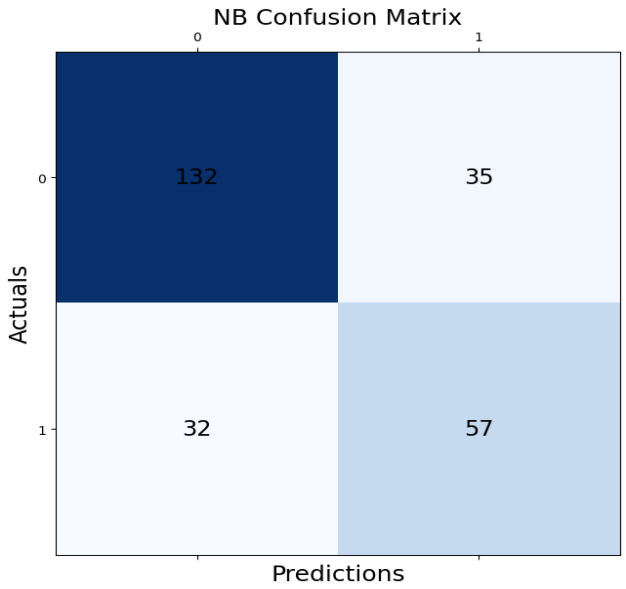
Pima Indian Diabetes Veri Seti NB algoritmasının bahsedilen özellikleriyle test edilmiştir.

```
In [11]: def StartNB(xtrain,ytrain,xtest,ytest):
             kFold = StratifiedKFold(n_splits=5,random_state=1,shuffle=True)
             cv_results = cross_val_score(estimator=GaussianNB(),X=xtrain,y=ytrain,cv=kFold,scoring='accuracy')
             print('%s: %f (%f)' %('NB',cv_results.mean(),cv_results.std()))
             clf = GaussianNB()
             clf = clf.fit(xtrain,ytrain)
             return clf.predict(xtest)
```

Şekil 3.4.4.1: NB sınıflandırma modelinin kullanımı

```
nb_Predict = StartNB(X_train,y_train,X_test,y_test)
ShowScores(nb_Predict,y_test,'NB')
ShowClassificaionReport(y_test,nb_Predict,'NB')
ShowConfussionMatrix(y_test,nb_Predict,'NB')
```

Şekil 3.4.4.2: NB modeli sonuç fonksiyonlarının çalıştırılması



Şekil 3.4.4.3: NB karmaşıklık matrisi

.

```
NB: 0.767542 (0.025390)
NB Accuracy: 0.73828125
NB Precision: 0.6195652173913043
NB Recall: 0.6404494382022472
NB F1 Score: 0.6298342541436465


NB

              precision    recall  f1-score   support

         0.0       0.80      0.79      0.80       167
         1.0       0.62      0.64      0.63        89

    accuracy                           0.74       256
   macro avg       0.71      0.72      0.71       256
weighted avg       0.74      0.74      0.74       256
```

Şekil 3.4.4.4: NB modeli ile yapılan sınıflandırma sonuçları

3.4.5 Support Vector Machine

SVM (Support Vector Machine), makine öğrenmesinde yaygın olarak kullanılan bir sınıflandırma ve regresyon algoritmasıdır. SVM, özellikle doğrusal olarak ayrılabilir olmayan veri setleri üzerinde yüksek performans gösteren bir algoritmadır.

SVM, bir sınıflandırma problemi çözerken, veri noktalarını bir uzayda temsil eder ve bu uzayda sınıfları birbirinden ayırmaya çalışır. Temel amacı, veri noktalarını sınıflar arasında maksimum marj ile ayırmaktır. Marj, sınıflar arasındaki en geniş boşluğu ifade eder ve bu boşlukta bulunan veri noktaları, karar sınırlarının belirlenmesinde önemli bir role sahiptir.

SVM, lineer olarak ayrılabilir veri setlerinde bir doğrusal sınıflandırıcı olarak çalışır. Ancak, çoğu gerçek dünya veri seti lineer olarak ayrılabilir değildir. Bu durumda, SVM çekirdek fonksiyonları kullanarak verileri daha yüksek boyutlu bir uzaya yansıtır ve lineer olarak ayrılabilir hale getirir. Bu sayede, verilerin daha karmaşık yapılarını modelleyebilir ve doğru sınıflandırma yapabilir.

SVM'nin avantajlarından biri, karar sınırlarının yanı sıra sınıf etiketlerine olan duyarlılığının az olmasıdır. Bu, SVM'nin aykırı değerlere veya gürültülü verilere daha dayanıklı olmasını sağlar. Ayrıca, SVM, doğrusal olmayan sınıflandırma problemlerinde yüksek esneklik sunar.

SVM'nin bir diğer önemli özelliği, destek vektörleri kullanmasıdır. Destek vektörler, karar sınırlarına en yakın veri noktalarıdır ve sınıflandırma sürecinde belirleyici bir rol oynarlar. Bu nedenle, SVM, sınıflandırma modelini destek vektörler üzerinden temsil eder.

SVM, sınıflandırma yanı sıra regresyon problemlerinde de kullanılabilir. Regresyon için SVM, veri noktalarını hedef değerlere en uygun bir şekilde sınıflandırır.

SVM, doğruluk oranı yüksek, iyi genelleme yapabilen ve veriler arasındaki ilişkileri yakalamak için esnek bir algoritmadır. Ancak, SVM'nin bazı dezavantajları vardır. Özellikle, büyük veri setlerinde ve yüksek boyutlu veri uzaylarında hesaplama maliyeti yüksek olabilir. Ayrıca, SVM'nin hiperparametrelerinin ayarlanması ve doğru bir şekilde seçilmesi önemlidir.

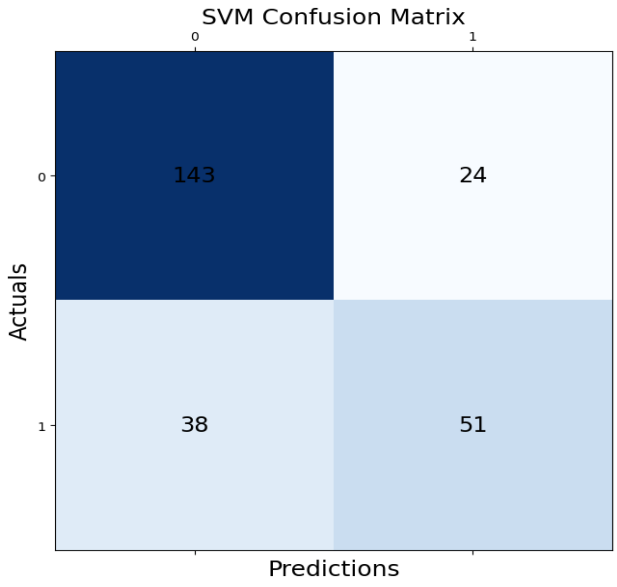
Pima Indian Diabetes Veri Seti, SVM algoritmasının bahsedilen özellikleriyle test edilmiştir.

```
In [12]: def StartSVM(xtrain,ytrain,xtest,ytest):
             kFold = StratifiedKFold(n_splits=5,random_state=1,shuffle=True)
             cv_results = cross_val_score(estimator=SVC(gamma='auto'),X=xtrain,y=ytrain,cv=kFold,scoring='accuracy')
             print('%s: %f (%f)' %('SVM',cv_results.mean(),cv_results.std()))
             clf = SVC(gamma='auto')
             clf = clf.fit(xtrain,ytrain)
             return clf.predict(xtest)
```

Şekil 3.4.5.1: SVM sınıflandırma modelinin kullanımı

```
svm_Predict = StartSVM(X_train,y_train,X_test,y_test)
ShowScores(svm_Predict,y_test,'SVM')
ShowClassificaionReport(y_test,svm_Predict,'SVM')
ShowConfussionMatrix(y_test,svm_Predict,'SVM')
```

Şekil 3.4.5.2: SVM modeli sonuç fonksiyonlarının çalıştırılması



Şekil 3.4.5.3: SVM karmaşıklık matrisi

```
SVM: 0.787207 (0.026412)
SVM Accuracy: 0.7578125
SVM Precision: 0.68
SVM Recall: 0.5730337078651685
SVM F1 Score: 0.6219512195121951


SVM

              precision    recall  f1-score   support

         0.0       0.79      0.86      0.82       167
         1.0       0.68      0.57      0.62        89

    accuracy                           0.76       256
   macro avg       0.74      0.71      0.72       256
weighted avg       0.75      0.76      0.75       256
```

Şekil 3.4.5.4: SVM modeli sınıflandırma sonuçları.

3.4.6 Decission Trees

Sınıflandırma ve regresyon problemlerinde kullanılan bir karar ağacı algoritmasıdır. Karar ağaçları, veri setinin özelliklerine dayanarak sınıflandırma veya regresyon yapmak için kullanılan grafiksel bir yapıdır.

DT, bir özellik üzerindeki değerlerin belirli bir eşik değeriyle ayrılması yoluyla veri setini bölerek ağacı oluşturur. Her düğümde, bir özellik ve bir eşik değeri seçilerek veri seti ikiye ayrılır. Bu süreç, her bölünmüş veri kümesi için rekürsif olarak tekrarlanır. Bölmeler, belirlenen bir durma kriterine veya bir bölünmenin belirli bir minimum veri noktası sayısına ulaşmasına kadar devam eder.

DT algoritması, sınıflandırma problemlerinde her bölünmüş veri kümesi için Gini indeksi adı verilen bir ölçüm kullanır. Gini indeksi, bir veri kümesindeki farklı sınıflara ait örneklerin karışıklığını ölçer. Ağacı oluştururken, en düşük Gini indeksine sahip bölme noktası seçilir. Bu şekilde, homojen sınıflardan oluşan yaprak düğümleri elde edilir.

Regresyon problemlerinde ise CART, her bölünmüş veri kümesi için hedef değişkenin varyansını minimize etmeye çalışır. Bölme noktası, veri kümesinin en düşük varyansa sahip olduğu değeri seçer.

DT algoritması, sınıflandırma ve regresyon problemlerinde kullanılan basit ve etkili bir yöntemdir. Karar ağaçları, veri setini anlamak ve yorumlamak açısından da kolaylık sağlar. Ancak, aşırı uyum (overfitting) sorunuyla karşılaşabilir ve daha karmaşık yapıları modellemekte sınırlı olabilir. Bu nedenle, aşırı uydurmayı engellemek için ağaçların derinliği veya minimum örnekleme sayısı gibi parametrelerin kontrol edilmesi önemlidir.

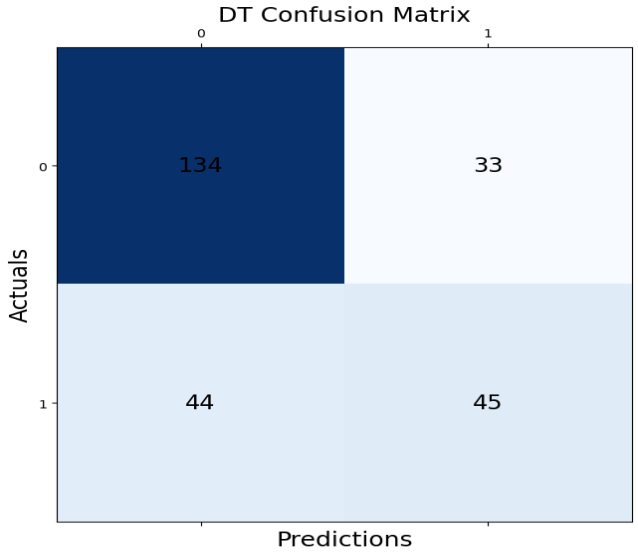
Pima Indian Diabetes Veri Seti, DT algoritmasının bahsedilen özellikleriyle test edilmiştir.

```
In [10]: def StartDT(xtrain,ytrain,xtest,ytest):
             kFold = StratifiedKFold(n_splits=5,random_state=1,shuffle=True)
             cv_results = cross_val_score(estimator=DecisionTreeClassifier(),X=xtrain,y=ytrain,cv=kFold,scoring='accuracy')
             print('%s: %f (%f)' %('DT',cv_results.mean(),cv_results.std()))
             clf = DecisionTreeClassifier()
             clf = clf.fit(xtrain,ytrain)
             return clf.predict(xtest)
```

Şekil 3.4.6.1: DT sınıflandırma modelinin kullanımı

```
dt_Predict =StartDT(X_train,y_train,X_test,y_test)
ShowScores(dt_Predict,y_test,'DT')
ShowClassificaionReport(y_test,dt_Predict,'DT')
ShowConfussionMatrix(y_test,dt_Predict,'DT')
```

Şekil 3.4.6.2: DT modeli sonuç fonksiyonlarının çalıştırılması



Şekil 3.4.6.3: DT karmaşıklık matrisi

```
DT: 0.687455 (0.059362)
DT Accuracy: 0.69921875
DT Precision: 0.573170731707317
DT Recall: 0.5280898876404494
DT F1 Score: 0.5497076023391814


DT

              precision    recall  f1-score   support

         0.0       0.76      0.79      0.77       167
         1.0       0.57      0.53      0.55        89

    accuracy                           0.70       256
   macro avg       0.67      0.66      0.66       256
weighted avg       0.69      0.70      0.70       256
```

Şekil 3.4.6.4: DT modeli ile yapılan sınıflandırma sonuçları

3.4.7 Random Forest Classifier

Sınıflandırma problemlerinde kullanılan bir makine öğrenmesi algoritmasıdır. RFC, birden fazla karar ağacını bir araya getirilmesiyle oluşturulan bir algoritmadır.

Random Forest, her bir karar ağacının veri setinin farklı alt kümeleriyle eğitildiği bir "karar ağacı ailesi" olarak düşünülebilir. RFC, her ağacın rastgele örnekler ve rastgele özelliklerle eğitildiği bir öğrenme süreci kullanır. Bu şekilde, farklı perspektiflerden ve farklı özelliklerle eğitilen birçok karar ağacı elde edilir.

RFC, her bir karar ağacının tahmin yapmasının ardından, sınıflandırma problemlerinde oylama yöntemi veya regresyon problemlerinde ortalama yöntemi kullanılarak sonuçların birleştirildiği bir ansambldir. Oylama yöntemiyle, sınıflandırma problemlerinde en çok oyu alan sınıf tahmini seçilir.

Random Forest, veri setindeki gürültüyü azaltabilir, aşırı uyuma (overfitting) karşı daha dirençli olabilir ve genel olarak iyi bir performans sergileyebilir. Ayrıca, RFC, özellik önem sıralamasını elde etmek için kullanılabilir. Her bir özellik, RFC tarafından yapılan ağaçların performansına dayanarak değerlendirilebilir.

RFC, sınıflandırma problemlerinde yaygın olarak kullanılan bir algoritmadır ve geniş bir uygulama yelpazesine sahiptir. Ancak, daha karmaşık yapıları modellemekte sınırlı olabilir ve büyük veri setlerinde hesaplama açısından maliyetli olabilir.

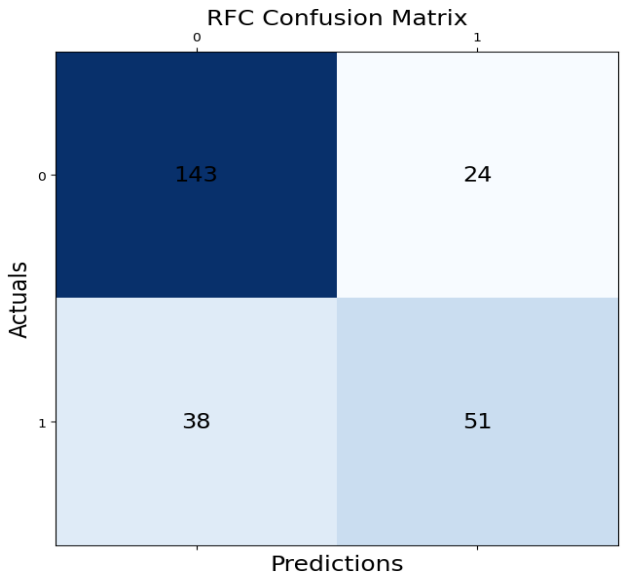
Pima Indian Diabetes Veri Seti, RFC algoritmasının yukarıda belirtilen özellikleriyle test edilmiştir.

```
In [13]: def StartRFC(xtrain,ytrain,xtest,ytest):
             kFold = StratifiedKFold(n_splits=5,random_state=1,shuffle=True)
             cv_results = cross_val_score(estimator=RandomForestClassifier(),X=xtrain,y=ytrain,cv=kFold,scoring='accuracy')
             print('%s: %f (%f)' %('RFC',cv_results.mean(),cv_results.std()))
             clf = RandomForestClassifier(random_state=46)
             clf = clf.fit(xtrain,ytrain)
             return clf.predict(xtest)
```

Şekil 3.4.7.1: RFC sınıflandırma modelinin kullanımı

```
rfc_Predict = StartRFC(X_train,y_train,X_test,y_test)
ShowScores(rfc_Predict,y_test,'RFC')
ShowClassificaionReport(y_test,rfc_Predict,'RFC')
ShowConfussionMatrix(y_test,rfc_Predict,'RFC')
```

Şekil 3.4.7.2: RFC modeli sonuç fonksiyonlarının çalıştırılması



Şekil 3.4.7.3: RFC karmaşıklık matrisi

```
RFC: 0.755911 (0.026470)
RFC Accuracy: 0.74609375
RFC Precision: 0.65
RFC Recall: 0.5842696629213483
RFC F1 Score: 0.6153846153846154


RFC

              precision    recall  f1-score   support

         0.0       0.79      0.83      0.81       167
         1.0       0.65      0.58      0.62        89

    accuracy                           0.75       256
   macro avg       0.72      0.71      0.71       256
weighted avg       0.74      0.75      0.74       256
```

Şekil 3.4.7.4: RFC modeli ile yapılan sınıflandırma sonuçları

4 Sonuçlar ve Tartışmalar

Bu çalışmada diyabeti tahmin etmek için çeşitli sınıflandırma tekniklerinden yararlanıldı. Önerilen yaklaşım, Gebelik, Glukoz, Kan Basıncı, Deri Kalınlığı, İnsülin, Vücut Kitle İndeksi, Soy Ağacı ve Yaş gibi özellikleri kullanarak KNN, NB, DT, SVM ve RF gibi farklı sınıflandırma algoritmalarını içermektedir

4.1 Değerlendirme Ölçütü

> "*Farklı makine öğrenimi algoritmalarının ürettiği sonuçların kalitesi, doğruluk, kesinlik, hatırlama ve F1-puanı değeri açısından değerlendirilir.* (Sokolova et al., 2006)"

Çalışmada, her sınıflandırma algoritması için doğruluk, F1-puanı, hatırlama ve kesinlik ölçümleri karışıklık matrisi kullanarak ölçüldü.

> "*Makine öğreniminde karışıklık matrisi, algoritmanın performansını göstermek için kullanılan bir tablodur. Performans, kullanıcı tarafından verilen giriş veri setini test ederek belirlenir. Aşağıdaki tablo, tahmin edilen ve gerçek değerleri göstermektedir.* (Yağanoğlu ve Köse, 2018)"

TP- Tahmin edilen değer pozitiftir ve doğrudur
TN- Tahmin edilen değer negatiftir ve doğrudur
FP - Tahmin edilen değer pozitiftir ve yanlıştır
FN- Tahmin edilen değer negatiftir ve yanlıştır

		Actual values	
		Positive (1)	Negative (0)
Forecasted values	Positive (1)	TP	FP
	Negative (0)	FN	TN

$$\textbf{Accuracy} = \frac{TP+FN}{TP+TN+FP+FN} \quad (1)$$

$$\textbf{Recall} = \frac{TP}{TP+FN} \quad (2)$$

$$\textbf{Precision} = \frac{TP}{TP+FP} \quad (3)$$

$$\textbf{F1-score} = \frac{2*Recall*Precision}{Recall+Precision} \quad (4)$$

Tablo 4.1.1: Karışıklık Matrisi Açıklama

4.2 Sonuç

Tablo 4.3.1, mevcut tüm özellikleri kullanan farklı sınıflandırma tekniklerinin sonuçlarını göstermektedir; KNN sınıflandırıcının, k=11 baz alındığında en iyi sonucu verdiği görülmektedir, %76.95 doğruluk, %71.42 kesinlik, %56 hatırlama ve %62 F1-puanı elde edilmiştir

Classification Technique	Accuracy	Precision	Recall	F1 score
KNN	**76,95%**	**71,42%**	**56,00%**	**62,00%**
DT	70,31%	57,47%	56,00%	56,00%
NB	73,00%	61,00%	64,00%	62,00%
SVM	75,00%	68,00%	57,00%	62,00%
RFC	74,00%	65,00%	58,00%	51,00%

Tablo 4.3.1: Kullanılan Sınıflandırma Tekniklerinin Karşılaştırılması

Yapay zekanın bir alt dalı olan Makine Öğrenimi, diyabet riski tahminini ve erken teşhisini tamamen değiştirme potansiyeline sahiptir. Diyabetin başarılı bir şekilde ele alınabilmesi için erken teşhis edilmesi gerekmektedir.

Bu çalışmanın temel amacı, Çeşitli Makine Öğrenimi Tekniklerini Kullanarak Diyabet Tahmini yapmak ve bu tekniklerin çıktı analizini gerçekleştirerek en yüksek doğruluğa sahip en iyi sınıflandırıcıyı bulmaktı ve bu amaca başarıyla ulaşıldı.

Bu çalışmada, çeşitli Makine Öğrenimi sınıflandırma tekniklerini test ettik. KNN algoritması, diğer Makine Öğrenimi sınıflandırma tekniklerine göre daha etkili ve daha iyi sonuçlar üretmektedir. KNN algoritmasının sınıflandırma doğruluğu %76,95'dir.

5 Kaynaklar

https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database - Pima Indians Diabetes Dataset

Robertson, G., Lehmann, E. D., Sandham, W., and Hamilton, D. (2011). Blood glucose prediction using artificial neural networks trained with the AIDA diabetes simulator: a proofof-concept pilot study. J. Electr. Comput. Eng.2011:681786. doi: 10.1155/2011/681786

Kang, Hyun. (2013). The prevention and handling of the missing data. Korean journal of anesthesiology.

Soni. M and Varma. S (2020), Diabetes Prediction using Machine Learning Techniques, International Journal of Engineering Research & Technology (IJERT)

Sokolova M., Japkowicz N., Szpakowicz S., (2006), Beyond Accuracy, F-score and ROC: a Family of Discriminant Measures for Performance Evaluation, American Association for Artificial Intelligence (www.aaai.org).

Yağanoğlu, M., & Köse, C., (2018), Real-time detection of important sounds with a wearable vibration based device for hearing-impaired people. Electronics, 7(4), 50.