Diyabet Risk Tahmini

İzzet Efe Demirci

Orijinal Veri Setinde KFold ile Model Çalışması

```
kf = StratifiedKFold(n_splits=5)
accuracies = []
confusion_matrices = []
kfold_results = ""

kul

for i, (train_index, test_index) in enumerate(kf.split(X, y),start=1):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    model_fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracies.append(accuracy_score(y_test, y_pred))
    conf = confusion_matrix(y_test, y_pred)
    confusion_matrices.append(conf)

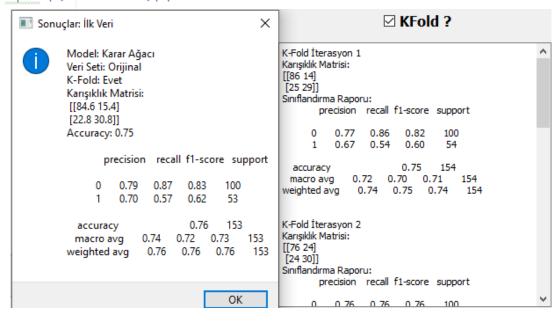
# Her iterasyon icin karışıklık matrisini ve sınıflandırma raporunu kfoldTextEdit alanına ekliyoruz
classification = classification_report(y_test, y_pred, labels=labels)
    kfold_results += f*K-Fold İterasyon {i}\nKarışıklık Matrisi:\n{conf}\nSınıflandırma Raporu:\n{classification}\n\n*
```

Orijinal veri seti için başarısı en yüksek olan Karar Ağacı sınıflandırıcısı kullanıldı.

Kfold çapraz doğrulama yöntemi ile veri seti 5 parçaya ayrılarak her iterasyonda veri setinin %20lik kısmı teste tabii tutuldu. Bu sonuçların ortalaması alınarak modelin ortalama başarısı ölçüldü.

```
# Sonuçlar1 kfoldTextEdit alanına yazdırıyoruz
self.kfoldTextEdit.setPlainText(kfold_results) # K-Fold sonuçlarını yazdırıyoruz
self.show_data_in_table(data)

cm = np.mean(confusion_matrices, axis=0)
accuracy = np.mean(accuracies)
```



Dengesizlik ile Başa Çıkma Deneysel Çalışması

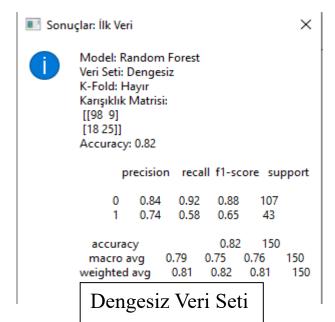
```
elif dataset_name == "Dengesiz":
    class_0 = original_data[original_data["Outcome"] == 0]
    class_1 = original_data[original_data["Outcome"] == 1]
    class_1_downsampled = resample( *arrays:class_1, replace=False, n_samples=len(class_0) // 2, random_state=42)
    imbalanced_data = pd.concat([class_0, class_1_downsampled])
    self.show_data_in_table(imbalanced_data)
    imbalanced_data.to_csv( path_or_buf: "imbalanced.csv",index=False)

smote = SMOTE(random_state=42)
    X_smote, y_smote = smote.fit_resample(imbalanced_data.drop(columns=["Outcome"]),imbalanced_data["Outcome"])
    balanced_data = pd.concat( objs: [pd.DataFrame(X_smote, columns=X.columns), pd.Series(y_smote, name="Outcome")], axis=1)
    balanced_data.to_csv( path_or_buf: "balanced.csv",index=False)

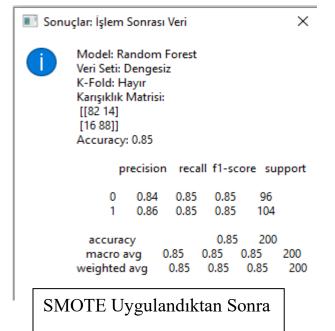
self.show_data_in_table(imbalanced_data)
```

Orijinal veri setini manipüle ederek dengesiz hale getiriyoruz. Dengesizlik ile SMOTE yöntemi ile başa çıkıyoruz.

SMOTE yöntemi, azınlık sınıfının sayısının çoğunluk sınıfı sayısına dengelemek için sentetik veri ekleme yöntemidir.



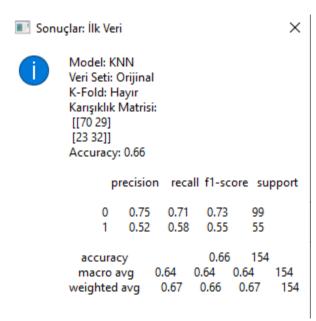
return imbalanced_data, balanced_data



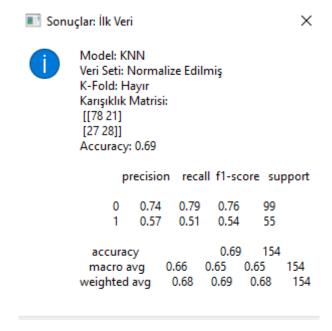
Normalizasyon ile Deneysel Çalışma

```
if dataset_name == "Normalize Edilmis":
    scaler = MinMaxScaler()
    X_normalized = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
    normalized_data = pd.concat( objs: [X_normalized, y], axis=1)
    normalized_data.to_csv( path_or_buf: "normalized.csv",index=False)
    self.show_data_in_table(normalized_data)
    return normalized_data, None
```

Orijinal veri setinde verileri minimum maximum normalizasyon ile 0-1 arasında değerlere normalize ediyoruz.



Orijinal Veri Seti



Normalize Edilmiş Veri Seti

Gürültülü Veriler ile Deneysel Çalışma

```
elif dataset_name == "Gürültülü":
    # Gürültü eklenecek örnek sayısını hesapla (toplam örnek sayısının %5'i)
    noise\_count = int(len(X) * 0.05)
    # Rastgele örnekler seç
    random_indices = np.random.choice(X.index, size=noise_count, replace=False)
    # Gürültü eklenecek örnekleri seç ve bu örneklere gürültü ekle
    X_noised = X.copy()
    for index in random_indices:
        noise = np.random.normal( loc: 0, scale: 0.1, X.loc[index].shape) # Her bir örnek için gürültü
        X_noised.loc[index] += noise # Gürültüyü ekle
    # Gürültü eklenmiş veri ve etiketleri birleştir
    noised_data = pd.concat( objs: [X_noised, y], axis=1)
    self.show_data_in_table(noised_data)
    noised_data.to_csv( path_or_buf: "noised.csv",index=False)
    # Gürültü eklenmiş olan verileri denoised_data'ya al
    X_{denoised} = X_{noised}[(np.abs(X_{noised} - X) < 0.1).all(axis=1)]
    y_denoised = y[X_denoised.index]
    denoised_data = pd.concat( objs: [X_denoised, y_denoised], axis=1)
    denoised_data.to_csv( path_or_buf: "denoised.csv",index=False)
    # Veriyi tablo halinde göster
    self.show_data_in_table(denoised_data)
    return noised_data, denoised_data
                                                                                            ×
                                           ×
                                                   Sonuçlar: İşlem Sonrası Veri
 Sonuçlar: İlk Veri
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	etesPedigreeFunc	Age	Outcome
1	6.0	148.0	72.0	35.0	0.0	33.6	0.627	50.0	1
2	1.0	85.0	66.0	29.0	0.0	26.6	0.351	31.0	0
3	8.0	183.0	64.0	0.0	0.0	23.3	0.672	32.0	1
4	1.0	89.0	66.0	23.0	94.0	28.1	0.167	21.0	0
5	-0.01936280229	136.8519235095	40.15319823157	35.02479406071	168.0516379191	43.10406871197	2	33.18745903924	1
6	5.0	116.0	74.0	0.0	0.0	25.6	0.201	30.0	0
7	3.0	78.0	50.0	32.0	88.0	31.0	0.248	26.0	1
8	10.0	115.0	0.0	0.0	0.0	35.3	0.134	29.0	0
9	2.0	197.0	70.0	45.0	543.0	30.5	0.158	53.0	1
<							1		

Yandaki kod ile orijinal veri setine gürültülü veriler ekliyoruz.

Ardından Gürültüleri çıkarıyoruz ve iki şekilde de modeli test edip sonuçlarına



Model: Lojistik Regresyon Veri Seti: Gürültülü K-Fold: Hayır Karışıklık Matrisi: [[78 21] [18 37]] Accuracy: 0.75

0 0.81 0.79 0.80 99 1 0.64 0.67 0.65 55 accuracy 0.75 154

0.73

weighted avg 0.75 0.75 0.75

macro avg

precision recall f1-score support

0.73

0.73

Model: Lojistik Regresyon Veri Seti: Gürültülü

> K-Fold: Hayır Karışıklık Matrisi: [[91 13]

[11 32]] Accuracy: 0.84

precision recall f1-score support

0 0.89 0.88 0.88 104 1 0.71 0.74 0.73 43

accuracy 0.84 147 macro avg 0.80 0.81 0.81 147 weighted avg 0.84 0.84 0.84 147

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	etesPedigreeFunc	Age	Outcome
1	6.0	148.0	72.0	35.0	0.0	33.6	0.627	50.0	1
2	1.0	85.0	66.0	29.0	0.0	26.6	0.351	31.0	0
3	8.0	183.0	64.0	0.0	0.0	23.3	0.672	32.0	1
4	1.0	89.0	66.0	23.0	94.0	28.1	0.167	21.0	0
5	0.0	137.0	40.0	35.0	168.0	43.1	2.288	33.0	1
6	5.0	116.0	74.0	0.0	0.0	25.6	0.201	30.0	0
7	3.0	78.0	50.0	32.0	88.0	31.0	0.248	26.0	1
8	10.0	115.0	0.0	0.0	0.0	35.3	0.134	29.0	0
9	2.0	197.0	70.0	45.0	543.0	30.5	0.158	53.0	1
/									

Predict Proba Metodu

```
elif model_name == "Karar Ağacı":
    model = DecisionTreeClassifier(max_depth=5)
X_train, _, y_train, _ = train_test_split( *arrays: X, data["Outcome"], test_size=0.2, random_state=None)
model.fit(X_train, y_train)
probabilities = model.predict_proba([user_input])[0]
# Sonuçları belirle
result_diyabet_var = probabilities[1] # Diyabet Var için olasılık
result_divabet_yok = probabilities[0] # Divabet Yok icin olasılık
# Kullanıcıya sonucu göster
result_message = (f"Diyabet Yok olasılığı: {result_diyabet_yok:.2f}\n"
                 f"Diyabet Var olasılığı: {result_diyabet_var:.2f}")
QMessageBox.information(self, "Tahmin Sonucu", result_message)
       Predict Proba metodu ile bir örneğin belirli bir
       sınıfa ait olma olasılığı tahmin edilir.
```

Ekrandaki girdileri verdiğimizde ve en iyi model olan karar ağacını kullandığımızda böyle bir tahminde bulundu.

Girilen verilere göre modelin tahmini, hastanın diyabetinin olmaması yönünde bir sonuç veriyor.

Pregnancies				
3	■ Tahm	in Sonucu	×	
Glucose				
150		Diyabet Yok olasılığı: 0.		
BloodPressure		Diyabet Var olasılığı: 0.3	olasiligi: 0.32	
100]	ОК		
SkinThickness			_	
36				
Insulin				
0				
BMI				
36				
DiabetesPedigreeFunction				
2				
Age		Tahmin Yap		
28				