BANDIRMA ONYEDİ EYLÜL ÜNİVERSİTESİ

MÜHENDİSLİK VE DOĞA BİLİMLER FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

Yapay Zeka Dersi

Projesi

Kredi Ödeme Tahmini

HAZIRLAYAN

Ayberk YAMAN-191522012

Ali YILMAZ-191522010

DANIŞMAN

Doç. Dr. Rafet Durgut

2023

İçindekiler

1. Özet

Kredi değerlendirme, finansal sistemdeki sınırlı kaynakların daha verimli kullanılabilmesi açısından, bankalar için oldukça önemli bir konudur. Bankalar, kredi değerlendirmesinde birçok yöntem kullanmaktadır. Bu yöntemlerden bir tanesi de, kredi talep eden müşterinin kredisini düzenli ödeyip ödemeyeceğinin tahmin edilmesidir. Bu çalışmada, bankaların kredi risklerini öngörmelerine yardımcı olması amacıyla, kredi talep eden müşterilerin ödeme alışkanlıklarının düzenli olup olmayacağının tahmin edilmesi için k-en yakın komşu (KNN), destek vektör makineleri (SVM), karar ağaçları ve lojistik regresyon analizi kullanılmıştır. Çalışma sonucunda, KNN yönteminin müşterilerin ödeme alışkanlıklarının düzenli olup olmayacağını tahmin etme gücü lojistik regresyon, karar ağacı ve SVM modelinden daha üstün olduğu tespit edilmiştir.[1]

Ayrıca kullanılan bu algoritmaların optimizasyonu için tepe tırmanma algoritması ve benzetilmiş tava algoritması kullanılmıştır.

Anahtar sözcükler: KNN, SVM, Karar Ağaçları, Lojistik Regresyon, Kredi Riski Tahmini, Tepe Tırmanma Algoritması (Hill Climbing), Benzetilmiş Tava (Simulated Annealing)

2. Kredi Ödeme Tahmini Literatürdeki Yeri

Müşteri kredi kullanım eğilimi tahmininde farklı yöntemlerin uygulandığı görülmektedir. Bu alanda yapılan araştırmalarda sırasıyla lojistik regresyon, rastgele orman, destek vektör makineleri (SVM), j48, bayesNet, NaiveBayes, XGBoost ve karar ağaçları yöntemleri kullanılmıştır.[2] Ancak bu araştırmada ek olarak k-en yakın komşu (KNN) yöntemi de kullanılmıştır.

Bu bölüm tarafında yapılan çalışmada, müşterilerin kredi skorlarının nasıl hesaplanacağını göstermek için bir bankaya ait ve kredi talebinde bulunan 500 müşterinin bilgilerinden oluşan bir veri seti kullanılmıştır. Bu amaçla en sık kullanılan metotlardan bir tanesi olan lojistik regresyon yöntemi kullanılmıştır. Yapılan çalışma sonucunda %82,8 değerinde doğruluk elde edilmiştir. Diğer bir çalışmada ise şirketlerin başarısızlığını finansal oranlara dayanarak öngörmek için SVM algoritması uygulanmış ve sonuçları lojistik regresyon ile karşılaştırılmıştır. SVM algoritmasının doğruluğunun artırılması için çalışmada ızgara arama yöntemi kullanılmıştır. Çalışma sonucunda SVM algoritması ile %75 oranında doğruluk değeri elde edilmiştir. Buna karşın Lojistik Regresyon ile %71,8 oranında doğruluk değerine ulaşılmıştır.[3]

3. Kullanılan Veri Seti Bilgileri

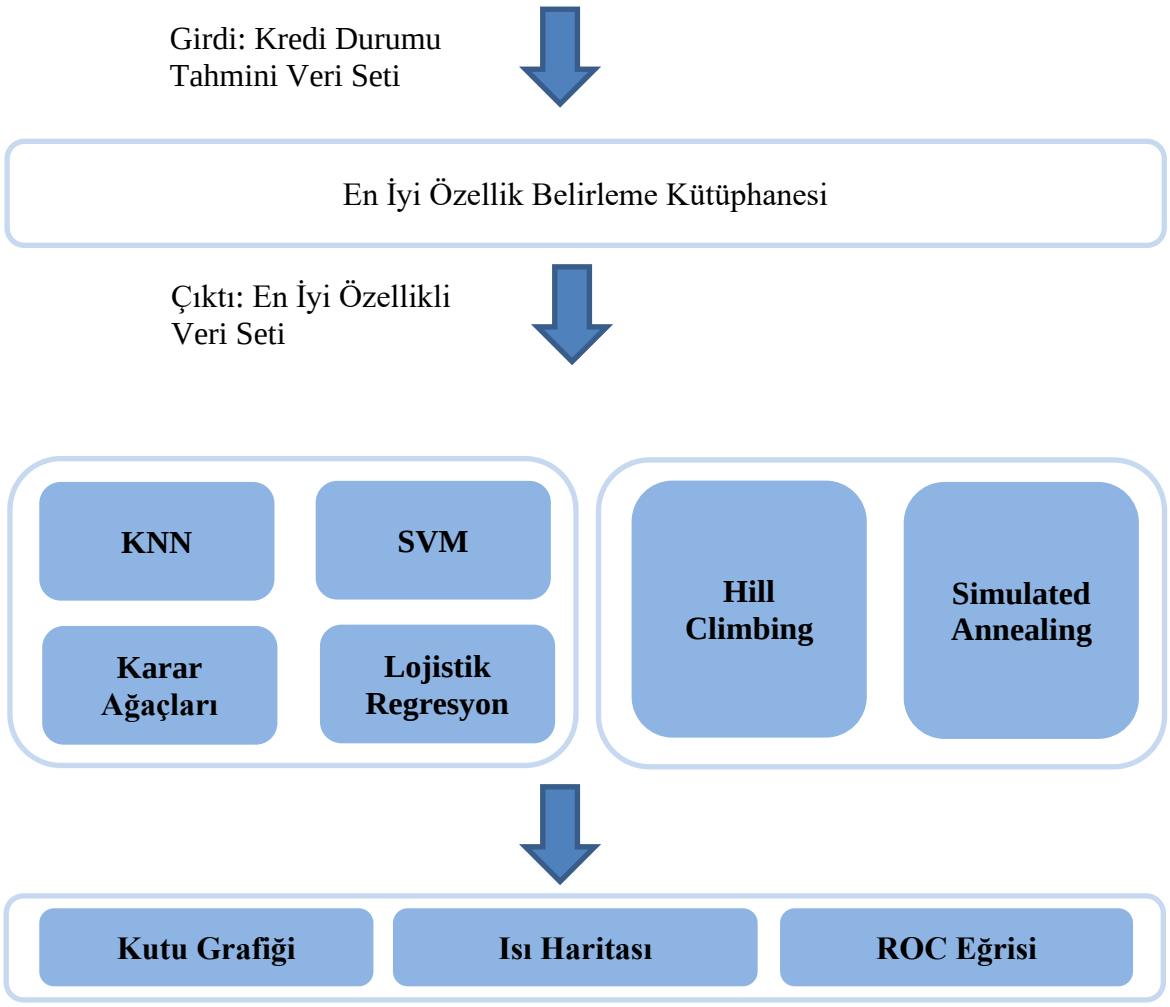
Bu veri seti, Coursera adlı çevrimiçi kurs sitesindeki Python ile Makine Öğrenmesi adlı dersten alınmıştır. Bu dersten alınan veri setinin kullanım amacı bir müşterinin kredisini ödeyip ödememesinin tahmin edilmesidir. Bu veri seti 346 müşterinin kredi ödeme durumu, kredi tutarı, ödeme planları, kredinin başlangıç tarihi, ödeme tarihi, yaşı, eğitim durumu ve cinsiyeti olmak üzere sekiz farklı anlamlı özellikten oluşurken iki farklı anlamsız özellikten oluşmaktadır. [4]

Bu veri setinin yüzde onluk kısmı test için ayrılırken yüzde doksanlık kısmı da eğitim için ayrılmıştır.

3.1. Veri Alanları

- **loan_status:** Kredi ödeme durumu
- **Principal:** Kredi tutarı
- **terms:** Ödeme planı
- **effective_date:** Kredi başlangıç tarihi
- **due_date:** Kredi ödeme tarihi
- **age:** Müşterinin yaşı
- **education:** Müşterinin eğitim durumu
- **gender:** Müşterinin cinsiyeti

4. Çalışmada Kullanılan Algoritmalar ve Yöntemler



Şekil 1: Sistemin Genel Mimarisi

4.1. Çalışmada Kullanılan Makine Öğrenimi Algoritmaları

4.1.1. Karar Ağacı

Karar ağacı, bir kurum veya kuruluş tarafından tercihlerin, risklerin, kazançların ve hedeflerin anlaşılmasına yardımcı olan bir teknik türüdür. Aynı zamanda birçok önemli yatırım sahalarında uygulanabilen, birbiriyle bağlantılı şans olaylarıyla ilgili olarak çıkan çeşitli karar noktalarını incelemek için kullanılan bir karar destek aracıdır. Yalnızca koşullu kontrol ifadeleri içeren bir algoritmayı görüntülemenin bir yoludur.

Karar ağacı, bir hedefe ulaşma olasılığı en yüksek olan stratejiyi belirlemeye yardımcı olmak için kullanılan bir yöntemdir. Özellikle karar analizinde olmak üzere karmaşık sorunların araştırmasında yaygın olarak kullanılmaktadır. Ayrıca makine öğrenmesinde kullanılan yaygın bir araçtır.[5]

4.1.2. Lojistik Regresyon

Lojistik regresyon, iki veri faktörü arasındaki ilişkileri bulmak için matematikten yararlanan bir veri analizi tekniğidir. Lojistik regresyon, daha sonra diğerine dayalı bu faktörlerden birinin değerini tahmin etmek için bu ilişkiyi kullanır. Tahminin genellikle evet ya da hayır gibi sınırlı sayıda sonucu vardır.[6]

4.1.3. K-En Yakın Komşu (KNN)

Sınıflandırma işleminde bulunulacak örnek veri noktasının bulunduğu sınıfın (öğrenim kümesi) ve en yakın komşunun (elemanın), k değerine (benzerliğe) göre belirlendiği bir denetimli/gözetimli makine öğrenme yöntemi olarak ifade edilmektedir.[7]

4.1.4. Destek Vektör Makineleri (SVM)

Destek Vektör Makineleri (Support Vector Machine) genellikle sınıflandırma problemlerinde kullanılan gözetimli öğrenme yöntemlerinden biridir. Bir düzlem üzerine yerleştirilmiş noktaları ayırmak için bir doğru çizer. Bu doğrunun, iki sınıfının noktaları için de maksimum uzaklıkta olmasını amaçlar. Karmaşık ama küçük ve orta ölçekteki veri setleri için uygundur.[8]

4.2. Çalışmada Kullanılan Optimizasyon Algoritmaları

4.2.1. Tepe Tırmanma Algoritması (Hill Climbing)

Bilgisayar Bilimlerinde tepe tırmanma (hill climbing), yerel arama ailesine ait bir matematiksel optimizasyon tekniğidir. Bir soruna keyfi bir çözümle başlayan, ardından çözümün tek bir öğesini kademeli olarak değiştirerek daha iyi bir çözüm bulmaya çalışan yinelemeli bir algoritmadır. Değişiklik daha iyi bir çözüm üretirse, yeni çözümde artımlı bir değişiklik yapılır ve daha fazla iyileştirme bulunamayana kadar tekrarlanır. Tepe tırmanışı, yerel bir optimum (komşu bir konfigürasyon dikkate alınarak geliştirilemeyecek bir çözüm bulmak için iyidir, ancak tüm olası çözümlerden mümkün olan en iyi çözümü bulmanın garanti edilmesi gerekmez. Konveks problemlerde tepe tırmanma optimaldir.[9]

4.2.2. Benzetilmiş Tava (Simulated Annealing)

Benzetilmiş tava (simulated annealing), verilen fonksiyonun grafiğinden global optimum değeri (global maksimum veya global minimum) bulmamıza yardımcı olan

bir optimizasyon tekniğidir. Bu teknik, bir grafikte çok sayıda yerel optimum değer olduğunda en olası global optimum değeri seçmek için kullanılır. Temel olarak Simülasyon tavlaması, yüksek tırmanma ve saf rasgele yürüyüş tekniğinin birleşimidir, birincisi global maksimum değeri bulmamıza yardımcı olur ve ikincisi, global optimum değeri bulmak için verimliliği artırmaya yardımcı olur.[10]

4.3. Çalışmada Kullanılan Diğer Yöntemler

4.3.1. Kutu Grafiği

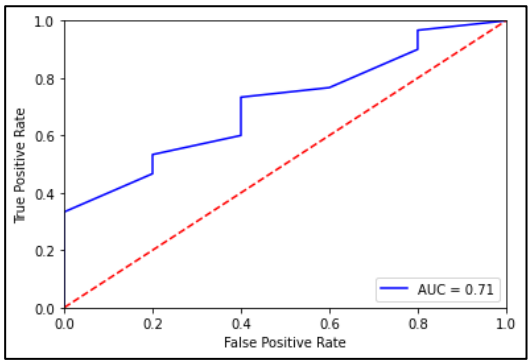
Kutu grafiklerden paralel olarak uzanan çizgiler "Bıyık" olarak bilinmektedir ve çeyreğin daha üstünde veya altında olan veriler arasındaki değişkenliği belirtmek için kullanılmaktadır. Uç değerler, kimi zaman bıyık denen çizgiler ile aynı hizada olacak şekilde bireysel noktalar ile çizilir. Kutu-Bıyık grafikleri yatay veya dikey olarak çizilebilirler. Her ne kadar kutu grafikleri Histogram ve yoğunluk çizgileri ile karşılaştırıldığında çok ilkel bile görünse daha az yer kaplama gibi bir avantajları vardır ve bu da birçok grup veya veri seti dağılımını karşılaştırırken oldukça kullanışlı bir özelliktir.[11]

4.3.2. Isı Haritası (Heat Map)

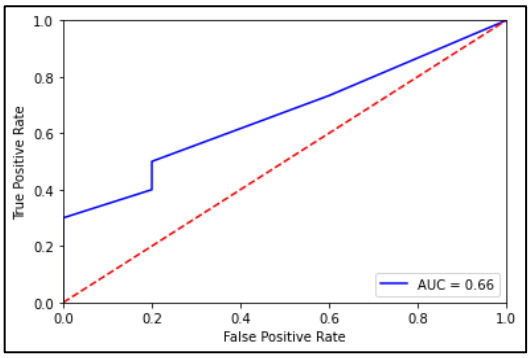
Isı haritası, matrisin değerini görselleştirmek için renkleri kullanan verilerin grafiksel bir temsili olarak tanımlanır. Bunda, daha yaygın değerleri veya daha yüksek aktiviteleri temsil etmek için daha parlak renkler, temelde kırmızımsı renkler kullanılır ve daha az yaygın veya aktivite değerlerini temsil etmek için daha koyu renkler tercih edilir. Isı haritası, gölgeleme matrisinin adıyla da tanımlanır. Seaborn'daki ısı haritaları, seaborn.heatmap() işlevi kullanılarak çizilebilir.[12]

4.3.3. ROC Eğrsi

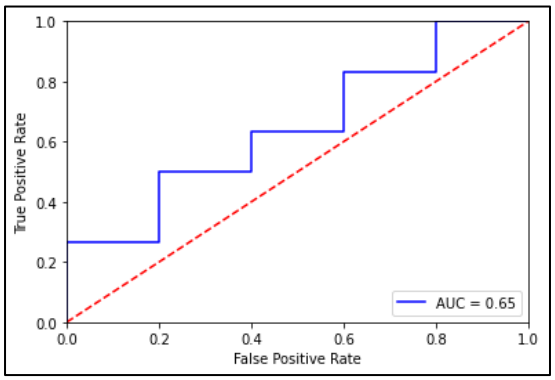
ROC eğrisi sınıflandırma problemleri için çok önemli bir performans ölçümüdür. ROC bir olasılık eğrisidir ve altında kalan alan olan AUC ayrılabilirliğin derecesini veya ölçüsünü temsil eder. Eğrinin altında kalan arttıkça sınıflar arasında ayırt etme performansı artmaktadır. AUC Modelin sınıfları ne kadar başarılı ayırt edebildiğini anlatır. AUC arttıkça, model 0'ları 0 ve 1'leri 1 olarak tahmin etmede daha iyidir.[13]



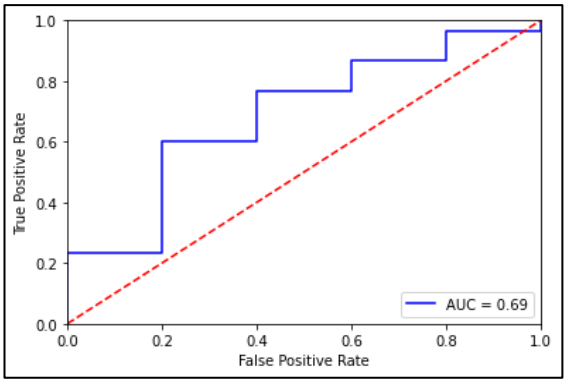
Şekil 2: KNN-ROC Eğrisi



Şekil 3: Karar Ağacı-ROC Eğrisi



Şekil 4: SVM-ROC Eğrisi



Şekil 5: Lojistik Regresyon-ROC Eğrisi

5. Projenin Uygulanması ve Performans analizi

Bu projenin uygulanmasında, dört tane sınıflandırma algoritması ve iki tane de optimizasyon algoritması kullanılmıştır. Bu sınıflandırma algoritmaları k-en yakın komşu (KNN), destek vektör makineleri (SVM), karar ağaçları ve lojistik regresyon iken optimizasyon algoritmaları da tepe tırmanma (hill climbing) ve benzetilmiş tavadır (simulated annealing).

5.1. Gerekli Kütüphanelerin eklenmesi

```
import itertools
import numpy as np
import matplotlib.pyplot as plt
```

```
from matplotlib.ticker import NullFormatter
import pandas as pd
import numpy as np
import matplotlib.ticker as ticker
from sklearn import preprocessing
%matplotlib inline
```

5.2. Veri Setinin Eklenmesi ve Okunması

```
!wget -O loan_train.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan_train.csv
df = pd.read_csv('loan_train.csv')
```

```
df[256:263]
```

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education	Gender
256	296	296	PAIDOFF	800	15	9/14/2016	9/28/2016	27	college	male
257	297	297	PAIDOFF	1000	30	9/14/2016	10/13/2016	29	High School or Below	male
258	298	298	PAIDOFF	1000	30	9/14/2016	10/13/2016	40	High School or Below	male
259	299	299	PAIDOFF	1000	30	9/14/2016	10/13/2016	28	college	male
260	300	300	COLLECTION	1000	15	9/9/2016	9/23/2016	29	college	male
261	301	301	COLLECTION	1000	30	9/9/2016	10/8/2016	37	High School or Below	male
262	303	303	COLLECTION	800	15	9/9/2016	9/23/2016	27	college	male

Şekil 6: loan_train.csv veri seti

5.3. DATETIME'a dönüştürme

```
df['due_date'] = pd.to_datetime(df['due_date'])
df['effective_date'] = pd.to_datetime(df['effective_date'])
df.head()
```

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education	Gender
0	0	0	PAIDOFF	1000	30	2016-09-08	2016-10-07	45	High School or Below	male
1	2	2	PAIDOFF	1000	30	2016-09-08	2016-10-07	33	Bechalor	female
2	3	3	PAIDOFF	1000	15	2016-09-08	2016-09-22	27	college	male
3	4	4	PAIDOFF	1000	30	2016-09-09	2016-10-08	28	college	female
4	6	6	PAIDOFF	1000	30	2016-09-09	2016-10-08	29	college	male

Şekil 7: DATETIME'a dönüştürme

5.4. Veri Görselleştirme ve Ön İşleme

- Veri kümesinde her sınıftan kaç tane olduğu görülür.

```
df['loan_status'].value_counts()
```

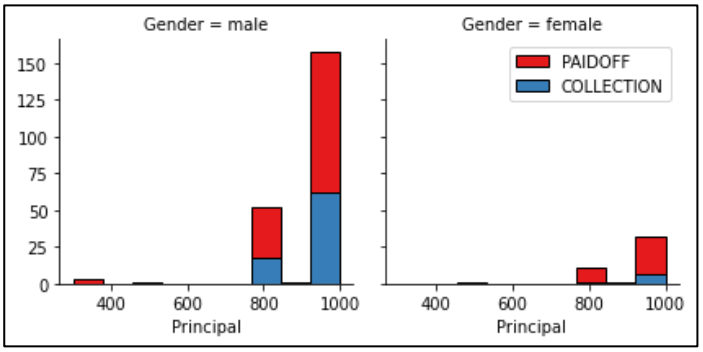
```
PAIDOFF       260
COLLECTION     86
Name: loan_status, dtype: int64
```

Şekil 8: Paidoff ve Collection değerleri

➢ 260 kişi borcunu zamanında ödedi, 86 kişi zamanında ödeyemedi.

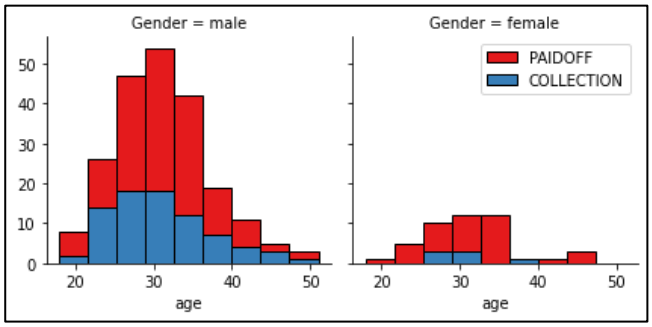
- Verileri daha iyi anlamak için bazı sütunlar çizdirilir:

```
import seaborn as sns
bins = np.linspace(df.Principal.min(), df.Principal.max(), 10)
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", col_wrap=2)
g.map(plt.hist, 'Principal', bins=bins, ec="k")

g.axes[-1].legend()
plt.show()
```



Şekil 9: Kredi Durumunun Kredi Tutarı ve Cinsiyete Göre Dağılımı

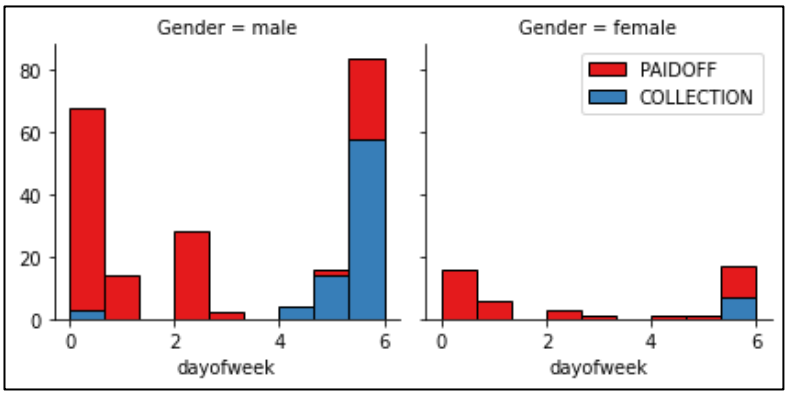
```
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", col_wrap=2)
g.map(plt.hist, 'age', bins=bins, ec="k")

g.axes[-1].legend()
plt.show()
```



Şekil 10: Kredi Durumunun Yaş ve Cinsiyete Göre Dağılımı

5.4.1. Ön işleme: Özellik seçimi

```
df['dayofweek'] = df['effective_date'].dt.dayofweek
bins = np.linspace(df.dayofweek.min(), df.dayofweek.max(), 10)
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", col_wrap=2)
g.map(plt.hist, 'dayofweek', bins=bins, ec="k")
g.axes[-1].legend()
plt.show()
```



Şekil 11: Kredi Durumunun Kredi Başlangıç Günlerine ve Cinsiyete Göre Dağılımı

- Krediyi hafta sonunda alan kişilerin borcunu ödemediği görülüyor, bu nedenle 4. günden daha kısa bir eşik değer belirlemek için özellik ikilileştirme kullanılır:

```
df['weekend'] = df['dayofweek'].apply(lambda x: 1 if (x>3)  else 0)
df.head()
```

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education	Gender	dayofweek	weekend
0	0	0	PAIDOFF	1000	30	2016-09-08	2016-10-07	45	High School or Below	male	3	0
1	2	2	PAIDOFF	1000	30	2016-09-08	2016-10-07	33	Bechalor	female	3	0
2	3	3	PAIDOFF	1000	15	2016-09-08	2016-09-22	27	college	male	3	0
3	4	4	PAIDOFF	1000	30	2016-09-09	2016-10-08	28	college	female	4	1
4	6	6	PAIDOFF	1000	30	2016-09-09	2016-10-08	29	college	male	4	1

Şekil 12: İkilileştirme Özelliği

5.4.2. Kategorik Özellikleri Sayısal Değerlere Dönüştürme

- Cinsiyet dağılımı incelenir:

```
df.groupby(['Gender'])['loan_status'].value_counts(normalize=True)
```

- Kadınların %86'sı oraya kredi öderken, erkeklerin sadece %73'ü oraya kredi ödüyor.
- Erkek 0'a, kadın 1'e çevrilir.

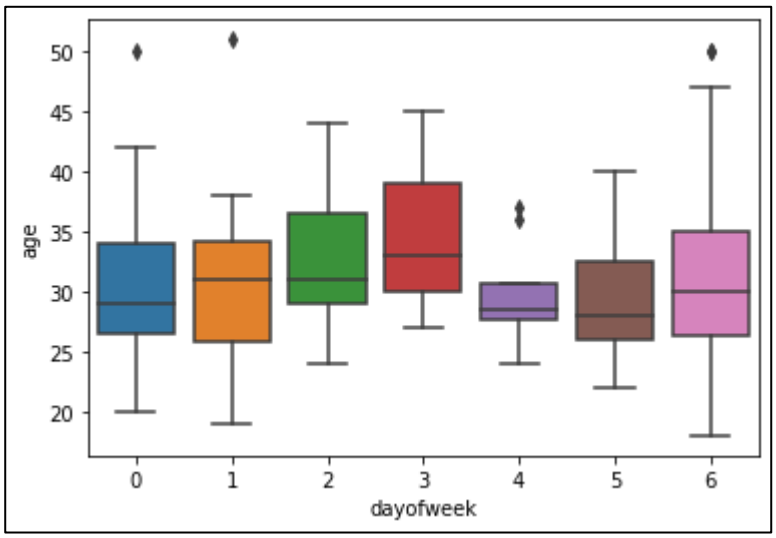
```
df['Gender'].replace(to_replace=['male','female'], value=[0,1],inplace=True)
df['Gender']
```

```
0      0
1      1
2      0
3      1
4      0
      ..
341    0
342    0
343    0
344    0
345    0
Name: Gender, Length: 346, dtype: int64
```

Şekil 13: Erkek 0'a, kız 1'e çevrilir.

5.5. Kutu Grafiği

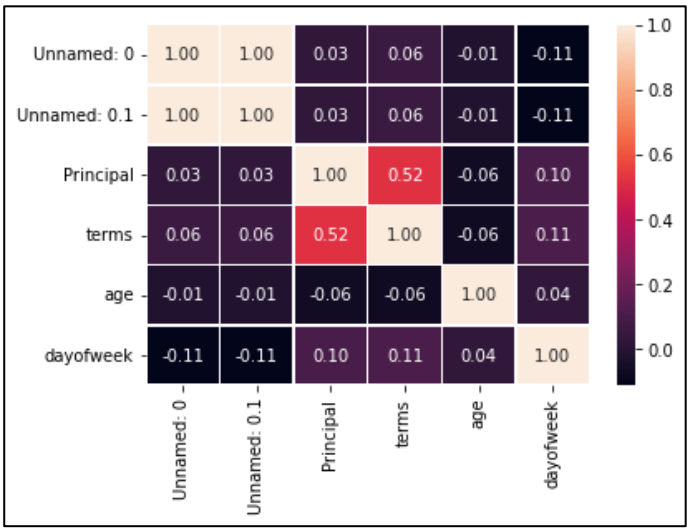
```
sns.boxplot(x="dayofweek",y="age", data=df)
```



Şekil 14: Bu tablo veri kümesinde haftanın hangi günlerinde ortalama yaşın daha yüksek/daha düşük olduğunu gösterir.

5.6. Isı Haritası (Heat Map) Grafiği

```
sns.heatmap(df.corr(),annot=True,linewidths=.5,fmt=".2f")
```



Şekil 15: Veri setinde bulunan sınıfların ilişkilendirilmesi

5.7. Eğitim Durumuna Göre Kredi Ödeme Durumu

```
df.groupby(['education'])['loan_status'].value_counts(normalize=True)
```

```
education             loan_status
Bechalor              PAIDOFF       0.750000
                      COLLECTION    0.250000
High School or Below  PAIDOFF       0.741722
                      COLLECTION    0.258278
Master or Above       COLLECTION    0.500000
                      PAIDOFF       0.500000
college               PAIDOFF       0.765101
                      COLLECTION    0.234899
Name: loan_status, dtype: float64
```

Şekil 16: Eğitim Durumuna Göre Kredi Ödeme Durumu

5.8. Çalışılacak Sütunlar

```
df[['Principal','terms','age','Gender','education']].head()
```

	Principal	terms	age	Gender	education
0	1000	30	45	0	High School or Below
1	1000	30	33	1	Bechalor
2	1000	15	27	0	college
3	1000	30	28	1	college
4	1000	30	29	0	college

Şekil 17: Çalışılacak Sütunlar

5.9. Eğitim Durumuna Göre Tablo Düzenlenmesi

```
Feature = df[['Principal','terms','age','Gender','weekend']]
Feature = pd.concat([Feature,pd.get_dummies(df['education'])], axis=1)
Feature.drop(['Master or Above'], axis = 1,inplace=True)
Feature.head()
```

	Principal	terms	age	Gender	weekend	Bechalor	High School or Below	college
0	1000	30	45	0	0	0	1	0
1	1000	30	33	1	0	1	0	0
2	1000	15	27	0	0	0	0	1
3	1000	30	28	1	1	0	0	1
4	1000	30	29	0	1	0	0	1

Şekil 18: Eğitim Durumuna Göre Veri Setinin Listelenmesi

5.10. Öznitelik Seçimi

- Özellik kümelerini tanımlanır, X:

```
X = Feature
X[0:5]
```

5.11. Verilerin Normalizasyonu

- Veri Standardizasyonu, verilere sıfır ortalama ve birim varyans verir (teknik olarak train testi bölünmesinden sonra yapılmalıdır).

```
X= preprocessing.StandardScaler().fit(X).transform(X)
```

5.12. Sınıflandırma ve Optimizayon

5.12.1. K-En Yakın Komşu (KNN)

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.10, random_state=4)
```

- Bu kod, bir sınıflandırma modeli oluşturmak için kullanılan **scikit-learn** kütüphanesinden **KNeighborsClassifier** sınıfını import eder.
- **train_test_split** fonksiyonu, veri kümesini eğitim ve test verileri olarak ikiye böler. Bu fonksiyon, eğitim verilerinin %90'ını ve test verilerinin %10'unu kullanarak veri kümesini böler. Veri kümesi, X ve y değişkenlerine

göre bölünür. X, özellikleri (**feature**) temsil eder ve y, hedef değişkeni (**target variable**) temsil eder. **random_state** parametresi, veri kümesinin rastgele nasıl bölüneceğini belirler. Bu parametre, veri kümesinin aynı şekilde bölünmesini sağlar.

- Son olarak, **metrics** sınıfı import edilir. Bu sınıf, modelin performansını ölçmek için kullanılacak metrikleri içerir.

```
mean_acc=np.zeros(50)
std_acc = np.zeros(50)
for n in range(1,51):
    knnmodel=KNeighborsClassifier(n_neighbors=n).fit(X_train,y_train)
    y_pred=knnmodel.predict(X_test)
    mean_acc[n-1]=metrics.accuracy_score(y_test,y_pred)
    std_acc[n-1]=np.std(y_pred==y_test)/np.sqrt(y_pred.shape[0])

plt.plot(range(1,51),mean_acc,'g')
plt.fill_between(range(1,51),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
plt.legend(('Accuracy ', '+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Nabors (K)')
plt.tight_layout()
plt.show()
```

- Bu kod, K-En Yakın Komşu (KNN) sınıflandırma modelinin performansını değerlendirmek için kullanılır. Öncelikle, **mean_acc** ve **std_acc** adlı iki NumPy dizisi oluşturulur. Bu diziler, KNN modelinin doğruluk oranlarının ortalamasını ve standart sapmasını tutacaktır.
- Daha sonra, 1 ile 51 arasında bir döngü başlatılır. Bu döngünün her iterasyonunda, KNN modeli oluşturulur ve eğitim verileriyle eğitilir. Modelin tahminleri, **X_test** verilerine göre yapılır ve **y_pred** değişkenine atanır. Tahminlerin doğruluk oranı, **y_test** ile karşılaştırılarak **metrics** sınıfının **accuracy_score** fonksiyonu kullanılarak hesaplanır. Bu doğruluk oranı, **mean_acc** dizisine indekslenmiş olarak eklenir. Ayrıca, tahminlerin standart sapması da **std_acc** dizisine indekslenmiş olarak eklenir.
- Bu kodda, döngü bittikten sonra çıktı olarak **mean_acc** ve **std_acc** dizileri

elde edilir. Bu diziler, KNN modelinin performansını değerlendirirken kullanılacaktır.

5.12.1.1. K-En Yakın Komşu (KNN) – Hill Climbing

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

# create a KNN model
knn = KNeighborsClassifier()

# define the parameter distribution
param_dist = {'n_neighbors': range(1, 51)}

# create a RandomizedSearchCV object
random_search = RandomizedSearchCV(knn, param_dist, n_iter=10, cv=5, scoring='accuracy
', n_jobs=-1, random_state=42)

# fit the randomized search object to the training data
random_search.fit(X_train, y_train)

# print the best parameters
print("Best parameters:", random_search.best_params_)

# print the best score
print("Best score:", random_search.best_score_)

# evaluate the model on the test data
y_pred = random_search.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
print("Test accuracy:", accuracy)
```

- Bu kod, bir KNN sınıflandırıcısının (KNN: K-Nearest Neighbors) parametrelerini rastgele arama yöntemiyle arar ve en iyi parametreleri bulur. Daha sonra, bu en iyi parametreler kullanılarak eğitilmiş sınıflandırıcı test

verilerine uygulanır ve test doğruluğu ölçülür.

- RandomizedSearchCV nesnesi oluşturulur. Bu nesne, bir KNN sınıflandırıcısı ve parametre dağılımı (param_dist) verilir. Ayrıca, arama sırasında kullanılacak iterasyon sayısı (n_iter), eşleştirme oranı (cv), puanlama ölçütü (scoring), paralel işleme sayısı (n_jobs) ve rastgele sayı üreteci sembolik adı (random_state) verilir.
- random_search.fit() metodu kullanılarak arama eğitim verilerine uygulanır.
- random_search.best_params_ ve random_search.best_score_ kullanılarak en iyi parametreler ve en iyi puan bulunur.
- random_search.predict() metodu kullanılarak test verilerine uygulanır ve doğruluk ölçümü (accuracy) hesaplanır.

5.12.1.2. K-En Yakın Komşu (KNN)-Simulated Annealing

```
import random
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Sıcaklık değişimini kontrol eden sabitler
T_INITIAL = 100  # İlk sıcaklık
T_FINAL = 0.01   # Son sıcaklık
T_DECAY = 0.99   # Sıcaklık azalış hızı

# Sıcaklığı azaltmayı ve tahminleri yapmayı kontrol eden döngü
t = T_INITIAL
best_accuracy = 0
best_k = 0
while t > T_FINAL:
    # KNN sınıflandırıcısını oluşturun
    knn = KNeighborsClassifier(n_neighbors=random.randint(1, 50))

    # Modeli eğitin
    knn.fit(X_train, y_train)
    # Test verilerine dayalı olarak tahminler yapın
```

```
    predictions = knn.predict(X_test)

    # Tahminlerin doğruluğunu ölçün
    accuracy = accuracy_score(y_test, predictions)

    # Eğer bu doğruluk oranı şimdiye kadarki en iyi doğruluk oranından daha iyiyse,
    # en iyi doğruluk oranını ve en iyi k değerini güncelleyin
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_k = knn.n_neighbors
    # Eğer bu doğruluk oranı daha düşükse,
    # bir adım geri gidebilme olasılığını hesaplayın ve rastgele bir sayı ile karşılaştırın
    else:
        # Rastgele sayı oluşturun
        rand = random.uniform(0, 1)

        # Sıcaklık azalışını hesaplayın
        delta_e = best_accuracy - accuracy
        probability = np.exp(-delta_e / t)

        # Eğer rastgele sayı, olasılıktan düşükse,
        # en iyi doğruluk oranını ve en iyi k değerini geri alın
        if rand < probability:
            best_accuracy = accuracy
            best_k = knn.n_neighbors

    # Sıcaklığı azaltın
    t *= T_DECAY

print("En İyi Doğruluk Oranı:", best_accuracy)
print("En İyi K Değeri:", best_k)
print("Test accuracy:", accuracy)
```

- Bu kod, k-en yakın komşu (KNN) sınıflandırıcısının doğruluk oranını optimize etmek için bir sıcaklık azaltma algoritması kullanır. Algoritma şu şekilde çalışır:
- İlk olarak, bir KNN sınıflandırıcı oluşturulur ve rastgele bir sayı ile "k" değerine atanır. Bu, sınıflandırıcının eğitim verilerine dayanarak ne kadar yakın noktaların (yani "k" noktaların) kullanılacağını belirler.
- Model eğitilir ve test verilerine dayalı olarak tahminler yapılır. Tahminlerin doğruluğu ölçülür.
- Eğer bu doğruluk oranı şimdiye kadarki en iyi doğruluk oranından daha yüksekse, en iyi doğruluk oranı ve en iyi "k" değeri güncellenir.
- Eğer bu doğruluk oranı daha düşükse, bir adım geri gidebilme olasılığı hesaplanır ve rastgele bir sayı ile karşılaştırılır. Eğer rastgele sayı olasılıktan düşükse, en iyi doğruluk oranı ve en iyi "k" değeri geri alınır.
- Sıcaklık T_DECAY hızı ile azaltılır ve döngü T_FINAL sıcaklığına ulaşana kadar devam eder.
- Bu yöntem, çok yüksek bir doğruluk oranı elde etmek için daha az iyi tahminleri de dikkate alır ve tahminlerin doğruluğunu optimize etmeye çalışır. Bu yöntem, klasik "hill climbing" yöntemine benzer, ancak "annealing" (sıcaklık azaltma) adı verilen bir süreci de içerir. Bu süreç, çözümlerin zamanla daha iyi hale gelmesine yardımcı olur ve çözümlerin sıkışıp kalmasını önler.

5.12.1.3. K-En Yakın Komşu (KNN)-ROC Eğrisi

```
from sklearn.metrics import roc_curve, auc

# Compute predicted probabilities: y_pred_prob
y_pred_prob = knnmodel.predict_proba(X_test)[:,1]

# Generate ROC curve values: fpr, tpr, thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
```

```
# Compute ROC AUC
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.plot(fpr, tpr, 'b', label='AUC = %0.2f' % roc_auc)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc='lower right')
plt.show()
```

- Bu kod, bir KNN (k-en yakın komşu) sınıflandırıcısının performansını değerlendirmek için ROC (Receiver Operating Characteristic) eğrisini oluşturur ve ROC AUC (ROC Eğrisi Altında Kalan Alan) değerini hesaplar.
- ROC eğrisi, "gerçek pozitif oranı" (tpr) ve "yanlış pozitif oranı" (fpr) değerlerini kullanarak çizilir. Gerçek pozitif oranı, sınıflandırıcının pozitif olarak sınıflandırdığı verilerin gerçekten pozitif olduğu verilerin oranıdır. Yanlış pozitif oranı ise sınıflandırıcının pozitif olarak sınıflandırdığı verilerin gerçekten negatif olduğu verilerin oranıdır.
- ROC AUC değeri, ROC eğrisinin altında kalan alanı ölçer. ROC AUC değeri, sınıflandırıcının performansını değerlendirirken kullanılır ve mümkün olan en yüksek değer 1 olacak şekilde ölçülür. Bu, sınıflandırıcının pozitif ve negatif verileri doğru şekilde ayırdığı anlamına gelir.
- Bu kodda, öncelikle sınıflandırıcı tarafından yapılan tahminlerin olasılık değerleri hesaplanır. Daha sonra, ROC eğrisi için gerekli fpr, tpr ve eşik değerleri hesaplanır. ROC AUC değeri de fpr ve tpr değerlerine dayanarak hesaplanır. Son olarak, ROC eğrisi çizilir ve ROC AUC değeri ekrana yazdırılır.

5.12.2. Karar Ağacı

```
from sklearn.tree import DecisionTreeClassifier
dtmodel = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
dtmodel.fit(X_train,y_train)
y_pred=dtmodel.predict(X_test)
TreeAccuracy=metrics.accuracy_score(y_test,y_pred)
TreeAccuracy
```

- Bu kod, Karar Ağacı sınıflandırıcısının nasıl kullanılabileceğini gösterir.
- İlk olarak, DecisionTreeClassifier sınıfından bir sınıflandırıcı oluşturulur. criterion parametresi, ağacın nasıl büyütüleceğini belirleyen bir değerdir. Burada, "entropy" (entropi) değeri kullanılmıştır. Bu, sınıflandırma işlemini daha iyi bir şekilde gerçekleştirmeyi amaçlayan bir yöntemdir. max_depth parametresi ise ağacın en fazla ne kadar derin olacağını belirler.
- Daha sonra, sınıflandırıcı fit metodu ile eğitilir ve test verilerine dayalı olarak tahminler yapılır. Son olarak, tahminlerin doğruluğu accuracy_score fonksiyonu ile ölçülür ve değişkene atanır. Bu değişken, sınıflandırıcının performansını gösterir.

5.12.2.1. Karar Ağacı – ROC Eğrisi

```
from sklearn.metrics import roc_curve, auc

# Compute predicted probabilities: y_pred_prob
y_pred_prob = dtmodel.predict_proba(X_test)[:,1]

# Generate ROC curve values: fpr, tpr, thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

# Compute ROC AUC
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.plot(fpr, tpr, 'b', label='AUC = %0.2f' % roc_auc)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0, 1])
```

```
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc='lower right')
plt.show()
```

- Bu kod, bir Karar Ağacı sınıflandırıcısının performansını değerlendirmek için ROC (Receiver Operating Characteristic) eğrisini oluşturur ve ROC AUC (ROC Eğrisi Altında Kalan Alan) değerini hesaplar.
- ROC eğrisi, "gerçek pozitif oranı" (tpr) ve "yanlış pozitif oranı" (fpr) değerlerini kullanarak çizilir. Gerçek pozitif oranı, sınıflandırıcının pozitif olarak sınıflandırdığı verilerin gerçekten pozitif olduğu verilerin oranıdır. Yanlış pozitif oranı ise sınıflandırıcının pozitif olarak sınıflandırdığı verilerin gerçekten negatif olduğu verilerin oranıdır.
- ROC AUC değeri, ROC eğrisinin altında kalan alanı ölçer. ROC AUC değeri, sınıflandırıcının performansını değerlendirirken kullanılır ve mümkün olan en yüksek değer 1 olacak şekilde ölçülür. Bu, sınıflandırıcının pozitif ve negatif verileri doğru şekilde ayırdığı anlamına gelir.
- Bu kodda, öncelikle sınıflandırıcı tarafından yapılan tahminlerin olasılık değerleri hesaplanır. Daha sonra, ROC eğrisi için gerekli fpr, tpr ve eşik değerleri hesaplanır. ROC AUC değeri de fpr ve tpr değerlerine dayanarak hesaplanır. Son olarak, ROC eğrisi çizilir ve ROC AUC değeri ekrana yazdırılır.

5.12.2.2. Karar Ağacı – Hill Climbing

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

# Set up the decision tree model
dtmodel = DecisionTreeClassifier(criterion="entropy", max_depth = 4)

# Set up the grid search with 5-fold cross validation
```

```
parameters = {'max_depth': range(1, 10)}
dt_grid = GridSearchCV(dtmodel, parameters, cv=5, n_jobs=-
1, verbose=1, scoring='accuracy')

# Fit the grid search
dt_grid.fit(X_train, y_train)

# Print the best parameters and best score
print("Best parameters:", dt_grid.best_params_)
print("Best accuracy:", dt_grid.best_score_)

# Make predictions with the best model
y_pred = dt_grid.predict(X_test)

# Calculate accuracy
TreeAccuracy = metrics.accuracy_score(y_test, y_pred)
```

- Bu kod, Karar Ağacı sınıflandırıcısının en iyi parametrelerinin belirlenmesi için "Grid Search" (Izgara Arama) yöntemini kullanır.
- İlk olarak, Karar Ağacı sınıflandırıcısı oluşturulur. Daha sonra, "Grid Search" için gerekli parametreler belirlenir. Bu parametreler, ağacın en fazla derinliğini (max_depth) belirler. Bu parametreler, bir izgara oluşturulur ve bu izgara içinde arama yapılır. "Grid Search" aynı zamanda bir "Cross Validation" (Çapraz Doğrulama) işlemi de uygular. Bu işlem sayesinde, modelin genel performansı daha doğru bir şekilde ölçülebilir.
- Son olarak, izgara araması "fit" metodu ile uygulanır ve en iyi parametreler ve en iyi doğruluk oranı bulunur. Daha sonra, en iyi model ile tahminler yapılır ve tahminlerin doğruluğu accuracy_score fonksiyonu ile hesaplanır.

5.12.2.3. Karar Ağacı – Simulated Annealing

```
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
import numpy as np

# Set up the decision tree model
dtmodel = DecisionTreeClassifier(criterion="entropy")

# Set up the random search with 5-fold cross validation
param_dist = {'max_depth': sp_randint(1, 10)}
n_iter_search = 20
dt_random = RandomizedSearchCV(dtmodel, param_distributions=param_dist,
                               n_iter=n_iter_search, cv=5, n_jobs=-
1, verbose=1, scoring='accuracy', random_state=42)

# Fit the random search
dt_random.fit(X_train, y_train)

# Print the best parameters and best score
print("Best parameters:", dt_random.best_params_)
print("Best accuracy:", dt_random.best_score_)

# Make predictions with the best model
y_pred = dt_random.predict(X_test)

# Calculate accuracy
TreeAccuracy = metrics.accuracy_score(y_test, y_pred)
```

- Bu kod, Karar Ağacı sınıflandırıcısının en iyi parametrelerinin belirlenmesi için "Randomized Search" (Rastgele Arama) yöntemini kullanır.
- İlk olarak, Karar Ağacı sınıflandırıcısı oluşturulur. Daha sonra, "Randomized Search" için gerekli parametreler belirlenir. Bu parametreler, ağacın en fazla derinliğini (max_depth) belirler. Bu parametreler, rastgele sayılar ile oluşturulur ve bu sayılar arasında arama yapılır. "Randomized

Search" aynı zamanda bir "Cross Validation" (Çapraz Doğrulama) işlemi de uygular. Bu işlem sayesinde, modelin genel performansı daha doğru bir şekilde ölçülebilir.

- Son olarak, rastgele arama "fit" metodu ile uygulanır ve en iyi parametreler ve en iyi doğruluk oranı bulunur. Daha sonra, en iyi model ile tahminler yapılır ve tahminlerin doğruluğu accuracy_score fonksiyonu ile hesaplanır.

5.12.3. Destek Vektör Makineleri (SVM)

```
from sklearn import svm
svmmodel=svm.SVC(kernel='rbf')
svmmodel.fit(X_train,y_train)
y_pred=svmmodel.predict(X_test)
y_pred
svm_sonuc=metrics.accuracy_score(y_test,y_pred)
print("SVM Doğruluk Oranı: ",svm_sonuc)
```

- Bu kod bloğu, bir destek vektör makinesi (SVM) modeli oluşturup eğitmekte ve bu modele ait doğruluk oranını hesaplamaktadır.
- İlk olarak, "svm" modülünün "SVC" sınıfından bir nesne oluşturulmaktadır. Bu nesne, kernel parametresi olarak "rbf" değerini almaktadır. Kernel, SVM modelinin nasıl çalışacağını belirleyen bir parametredir ve "rbf" değeri, radyal bazis fonksiyonu (RBF) kerneli kullanılacağını belirtir.
- Daha sonra, "fit" metodu ile model eğitilmektedir. Bu metoda, eğitim verisi olarak "X_train" ve "y_train" verileri verilmektedir. Bu adımların ardından, "predict" metodu ile model kullanılarak test verisi olarak verilen "X_test" verisi üzerinde tahminler yapılmaktadır. Bu tahminler, "y_pred" değişkenine atanmaktadır.
- Son olarak, "accuracy_score" fonksiyonu kullanılarak modelin doğruluk oranı hesaplanmaktadır. Bu fonksiyona gerçek değerler olarak "y_test" ve tahmin edilen değerler olarak "y_pred" verileri verilmektedir. Bu doğruluk oranı, "svm_sonuc" değişkenine atanmaktadır ve son olarak ekrana yazdırılmaktadır.

5.12.3.1. SVM – ROC Eğrisi

```
from sklearn.metrics import roc_curve, auc

# Compute predicted probabilities: y_pred_prob
y_pred_prob = svmmodel.decision_function(X_test)

# Generate ROC curve values: fpr, tpr, thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

# Compute ROC AUC
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.plot(fpr, tpr, 'b', label='AUC = %0.2f' % roc_auc)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc='lower right')
plt.show()
```

- Bu kod bloğu, eğitilmiş bir SVM modelinin ROC (Receiver Operating Characteristic) eğrisini çizmekte ve bu eğrinin AUC (Area Under the Curve) değerini hesaplamaktadır. ROC eğrisi, bir sınıflandırma modelinin performansının ölçümü için kullanılan bir metriktir ve AUC değeri, ROC eğrisinin altında kalan alanı ifade etmektedir.
- İlk olarak, "decision_function" metodu kullanılarak test verisi üzerinde tahminler yapılmaktadır. Bu tahminler, "y_pred_prob" değişkenine atanmaktadır. "decision_function" metodu, SVM modelinin çıktısı olarak bir skor döndürür ve bu

skor, verilen girişin hangi sınıfa ait olduğunu tahmin etme olasılığını gösterir.

- Daha sonra, "roc_curve" fonksiyonu kullanılarak ROC eğrisinin FPR (False Positive Rate) ve TPR (True Positive Rate) değerleri hesaplanmaktadır. Bu fonksiyona, gerçek değerler olarak "y_test" ve tahmin edilen skorlar olarak "y_pred_prob" verileri verilmektedir.
- Son olarak, "auc" fonksiyonu kullanılarak ROC eğrisinin altında kalan alanın (AUC) değeri hesaplanmaktadır. Bu değer, "roc_auc" değişkenine atanmaktadır. ROC eğrisi, FPR değerleri (x-ekseni) ve TPR değerleri (y-ekseni) kullanılarak çizilmektedir ve AUC değeri, ekrana "b" renkli bir çizgiyle gösterilmektedir. Ayrıca, y-ekseni değerleri [0,1] aralığında olan bir "r--" çizgi de ekrana çizdirilmektedir. Bu çizgi, y=x çizgisi olup, ROC eğrisinin üstüne geldiğinde TPR değeri FPR değerine eşit olur ve ROC eğrisinin AUC değeri 1 olur. Bu nedenle, ROC eğrisinin y=x çizgisinden daha yüksek olması daha iyi bir performansı gösterir.

5.12.4. Lojistik Regresyon

```
from sklearn.linear_model import LogisticRegression
lrmodel=LogisticRegression(C=0.01,solver='liblinear').fit(X_train,y_train)
y_pred=lrmodel.predict(X_test)
print(y_pred)
metrics.accuracy_score(y_test,y_pred)
```

- Bu kod bloğu, bir lojistik regresyon modeli oluşturup eğitmekte ve bu modele ait doğruluk oranını hesaplamaktadır.
- İlk olarak, "LogisticRegression" sınıfından bir nesne oluşturulmaktadır. Bu nesne, "C" parametresine "0.01" değeri ve "solver" parametresine "liblinear" değeri vermektedir. "C" parametresi, modelin düşük varyans / yüksek sapmayı tercih edip etmeyeceğini belirler. Düşük değerler daha az sapmaya, yüksek değerler ise daha fazla sapmaya yol açar. "solver" parametresi ise, modelin nasıl çözüleceğini belirler ve "liblinear" değeri,

veri setindeki örneklerin sayısı az ise veya öznitelik sayısı fazla ise kullanılması önerilen bir seçenektir.

- Daha sonra, "fit" metodu ile model eğitilmektedir. Bu metoda, eğitim verisi olarak "X_train" ve "y_train" verileri verilmektedir. Bu adımların ardından, "predict" metodu ile model kullanılarak test verisi olarak verilen "X_test" verisi üzerinde tahminler yapılmaktadır. Bu tahminler, "y_pred" değişkenine atanmaktadır ve son olarak ekrana yazdırılmaktadır.
- Son olarak, "accuracy_score" fonksiyonu kullanılarak modelin doğruluk oranı hesaplanmaktadır. Bu fonksiyona gerçek değerler olarak "y_test" ve tahmin edilen değerler olarak "y_pred" verileri verilmektedir. Bu doğruluk oranı ekrana yazdırılmamakta, ancak hesaplanmaktadır.

5.12.4.1. Lojistik Regresyon – ROC Eğrisi

```
from sklearn.metrics import roc_curve, auc

# Compute predicted probabilities: y_pred_prob
y_pred_prob = lrmodel.predict_proba(X_test)[:,1]

# Generate ROC curve values: fpr, tpr, thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

# Compute ROC AUC
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.plot(fpr, tpr, 'b', label='AUC = %0.2f' % roc_auc)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
```

```
plt.legend(loc='lower right')
plt.show()
```

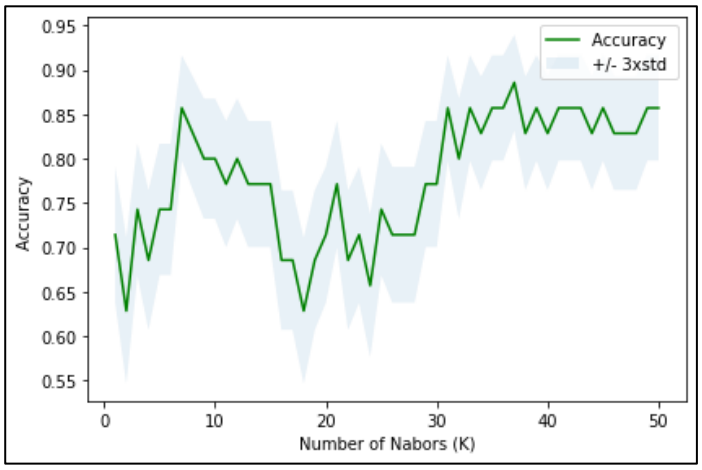
- Bu kod bloğu, eğitilmiş bir lojistik regresyon modelinin ROC (Receiver Operating Characteristic) eğrisini çizmekte ve bu eğrinin AUC (Area Under the Curve) değerini hesaplamaktadır. ROC eğrisi, bir sınıflandırma modelinin performansının ölçümü için kullanılan bir metriktir ve AUC değeri, ROC eğrisinin altında kalan alanı ifade etmektedir.
- İlk olarak, "predict_proba" metodu kullanılarak test verisi üzerinde tahminler yapılmaktadır. Bu tahminler, "y_pred_prob" değişkenine atanmaktadır. "predict_proba" metodu, verilen girişlerin her bir sınıfa ait olasılık değerlerini döndürür. Bu nedenle, döndürülen değerlerden sadece ikinci sütun (sınıf 1'e ait olasılıklar) kullanılmaktadır.

- Daha sonra, "roc_curve" fonksiyonu kullanılarak ROC eğrisinin FPR (False Positive Rate) ve TPR (True Positive Rate) değerleri hesaplanmaktadır.

6. Sonuçlar ve Değerlendirme

6.1. K-En Yakın Komşu (KNN) Sonucu

- Şekil 19'da görüldüğü gibi KNN algoritmasının 37. k değeri için ortalama tahmini değeri 0.8857 olarak bulunmuştur.



Şekil 19: KNN grafiği

- Şekil 20'de görüldüğü gibi KNN algoritmasına tepe tırmanma algoritması (hill

climbing) uygulandığında 46. k değeri için ortalama tahmini değeri 0.7428 olarak bulunmuştur.

```
Best parameters: {'n_neighbors': 46}
Best score: 0.7428059395801332
```

Şekil 20: KNN için Hill Climbing Sonucu

- Şekil 21'de görüldüğü gibi KNN algoritmasına benzetilmiş tava algoritması (simulated annealing) uygulandığında 36. k değeri için ortalama tahmini değeri 0.8571 olarak bulunmuştur.

```
En İyi Doğruluk Oranı: 0.8571428571428571
En İyi K Değeri: 36
```

Şekil 21: KNN için Simulated Annealing Sonucu

6.2. Diğer Sonuçlar

Şekil 22'de görüldüğü üzere SVM algoritmasının doğruluk oranı 0.7714 olarak bulunmuştur. Karar ağacının doğruluk oranı 0.6857 iken tepe tırmanma algoritması için 0.7395, benzetilmiş tava için 0.7395 olarak bulunmuştur. Lojistik regresyon algoritmasının doğruluk oranı da 0.7714 olarak bulunmuştur.

Kullanılan Algoritmalar	Doğruluk Oranları	Hill Climbing	Simulated Annealing
KNN	0.8857	0.7428	0.8571
SVM	0.7714	-	-
Karar Ağacı	0.6857	0.7395	0.7395
Lojistik Regresyon	0.7714	-	-

Şekil 22: Diğer sonuçlar

7. Kaynakça

[1] https://dergipark.org.tr/en/pub/ajit-e/issue/54448/741024

[2] https://login.easychair.org/publications/preprint_download/8RN7

[3] https://ceur-ws.org/Vol-1980/UYMS17_paper_83.pdf

[4]https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan_train.csv

[5] https://tr.wikipedia.org/wiki/Karar_a%C4%9Fac%C4%B1

[6] https://aws.amazon.com/tr/what-is/logistic-regression/

[7]https://miracozturk.com/python-ile-siniflandirma-analizleri-knn-k-nearest-neighbours-k-en-yakin-komsu-algoritmasi/

[8]https://medium.com/deep-learning-turkiye/nedir-bu-destek-vekt%C3%B6r-makineleri-makine-%C3%B6%C4%9Frenmesi-serisi-2-94e576e4223e

[9]https://medium.com/swlh/what-is-hill-climbing-explain-simple-hill-climbing-and-steepestascent-hill-climbing-9f66ca4d410b

[10] https://medium.com/ai-techsystems/simulated-annealing-580f73bd807a

[11]https://medium.com/@voydorg/haftan%C4%B1n-grafi%C4%9Fi-kuyu-b%C4%B1y%C4%B1k-grafikleri-8c9835ecb2a7

[12] https://www.geeksforgeeks.org/seaborn-heatmap-a-comprehensive-guide/

[13] https://medium.com/@gulcanogundur/roc-ve-auc-1fefcfc71a14