YAZ304
Sinir Ağları ve Makine Öğrenmesi Homework-1
Kaan İnce 210911216

Linear Regression

Verilen datasetini python veri dosyasında okundu. Nesne yönelimli programlama prosedürüne uymak açısından LinearR adında bir sınıf oluşturuldu. Bu sınıfa belirli methodlar eklendi.

Feature_Engineering Methodu

Bu fonksiyon da mpg veriseti *0.43 ile çarpıldı. Bulunan değer mileage ile çarpıldı. Ardından 100'e bölünerek litre adında bir feature oluşturuldu.

```
from re import S
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.preprocessing import MinMaxScaler, scale
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import r2_score ,mean_squared_error
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix
import math
import warnings
warnings.simplefilter(action='ignore', category=Warning)
data = pd.read_csv('cars_dataset.csv')
cars=data.copy()
class LinearR():
    def __init__(self,data):
        self.data=data
        self.X_train_new = None
    def feature_engineering(self,data):
        self.litre = data.insert(2, 'litre', pd.Series(self.data["mileage"]*(self.data["mpg"]*0.43))/100)
        self.litre = data["litre"]
        return self.litre
```

Data Visualization Methodu

Bu fonksiyonda cols listesine atılan mileage,tax,mpg ve engineSize featureları, price'e göre regplot tarafından görsel olarak ekrana getirildi. Boxplot ile fuelType ve price ekrana getirildi.

```
    def data_visualization(self):

        cols = ['mileage', 'tax', 'mpg', 'engineSize']
        plt.figure(figsize=(20,25))

        for i in range(len(cols)):
            plt.subplot(5,3,i+1)
            plt.title(cols[i] + ' - Price')
            sn.regplot(x=eval('data' + '.' + cols[i]), y=self.data["price"])

        plt.tight_layout()

        self.data['Make'].str.strip()
        plt.figure(figsize=(10,8))
        sn.boxplot(x=self.data["fuelType"], y=self.data["price"])
        plt.show()
```

Dummy Methodu

Bu fonksiyonda, transmission, fuelType, model kategorik verileri get_dummies methodu ile dönüştürüldü ve dataset güncellendi.

```
def dummy(self):
    dummies_list=['transmission','fuelType','model']
    for i in dummies_list:
        temp_df = pd.get_dummies(eval('cars' + '.' + i), drop_first=True)
        self.data = pd.concat([self.data, temp_df], axis=1)
        self.data.drop([i], axis=1, inplace=True)
    return self.data
```

Feature Delete, Train test ve Min Max Scaler Methodları

Feature_delete methodunda datasette olan make feature'ı datayı etkilemeyeceği için çıkarıldı. Train_test methodunda dataset, eğitmek ve test etmek için ayrıldı. Min_max_scaler methodunda ise year,enginSize,litre,tax ve price feature değerlerinin baskınlığını azaltmak için MinMaxScaler yöntemi uygulandı.

```
def feature_delete(self):
    self.data.drop(['Make'],axis=1,inplace=True)
def train_test(self):
    np.random.seed(0)
    self.df_train, self.df_test = train_test_split(self.data, train_size = 0.7, test_size = 0.3, random_state = 100)
def min_max_scaler(self):
    self.scaler = MinMaxScaler()
    scale_cols = ['year','engineSize','litre','tax','price']
    self.df_train[scale_cols] = self.scaler.fit_transform(self.df_train[scale_cols])
    self.y_train = self.df_train.pop('price')
    self.X_train = self.df_train
```

Linear Regression Methodu

Bu fonksiyonda, model oluşturuldu ve fit edildi. RFE ise datasette hedef değişkeni tahmin etmede etkili olması için oluşturuld

```
def linear_regression(self):
    self.lm = LinearRegression()
    self.lm.fit(self.X_train,self.y_train)
    rfe = RFE(self.lm, n_features_to_select=10)
    rfe = rfe.fit(self.X_train, self.y_train)
    self.X_train_rfe = self.X_train[self.X_train.columns[rfe.support_]]
```

OLS Build ve OLS Error Methodları

Ols_build methodunda ols tablosu çizildi. Modeli etkileyen featurelar bastırıldı ve P değerlerinin 0.05 ten küçük olmasına dikkat edildi.Ols_error methodunda ise hata terimlerinin grafiği çizildi.

```
def ols_build(self,X,y):
    X = sm.add_constant(X)
    self.lm = sm.OLS(y,X).fit()
    print(self.lm.summary())
    return X
def ols_error(self):
    self.lm = sm.OLS(self.y_train,self.X_train_new).fit()
    y_train_price = self.lm.predict(self.X_train_new)
    fig = plt.figure()
    sn.distplot((self.y_train - y_train_price), bins = 20)
    fig.suptitle('Error Terms', fontsize = 20)
    plt.xlabel('Errors', fontsize = 18)
    plt.show()
```

Regression Pred Test , R2 Score ve Coefficients Methodları

Regression_pred_test methodunda, belirli featurelar listeye atıldı. Bu featurelar ile pred ve testler ayarlandı. R2_score methodunda, y_test ve y_pred'e göre r-squared bulundu. Coefficients methodunda ise, OLS tablosunda bulunan değişkenlerin katsayıları ekrana getirildi.

```
def regression_pred_test(self):
    num_vars = ['year','engineSize','litre','tax','price']
    self.df_test[num_vars] = self.scaler.fit_transform(self.df_test[num_vars])
    self.y_test = self.df_test.pop('price')
    self.y_test = self.y_test.values.reshape(-1, 1)
    self.X_test = self.df_test
    self.X_train_new = self.X_train_new.drop('const',axis=1)
    self.X_test_new = self.X_test[self.X_train_new.columns]
    self.X_test_new = sm.add_constant(self.X_test_new)
    self.y_pred = self.lm.predict(self.X_test_new)
    self.y_pred = self.y_pred.values.reshape(-1, 1)
    return self.y_test,self.y_pred
def r2_score(self):
    return r2_score(self.y_test, self.y_pred)
def coefficients(self):
    self.X_train_final =self. X_train_new[['year', 'engineSize', 'litre',' 8 Series',' California', ' R8',' X7',' i3',' i8']]
    self.X_train_final.columns
    self.lr_final = LinearRegression()
    self.lr_final.fit(self.X_train_final, self.y_train)
    self.coefficient = pd.DataFrame(self.lr_final.coef_, index = ['year', 'engineSize', 'litre',' 8 Series',' California', ' R8',' X7',' i3',' i8'],
    columns=['Coefficient'])
    return self.coefficient.sort_values(by=['Coefficient'], ascending=False)
```

Rmse ve Pca Methodları

Rmse methodunda, y_test ve y_pred'in mean squared error'u bulundu. Ardından rmse hesaplandı. Bu şekilde hata payı öğrenildi. Pca methodunda , belirli değişkenler listeye atıldı. PCA'yı etkileyecek değer girildi ve model tekrardan oluşturulup fit edildi. Yeni PCA r-squared hesaplandı.

```
def rmse(self):
    self.mse = mean_squared_error(self.y_test, self.y_pred)
    self.rmse = math.sqrt(self.mse)
    return self.rmse
def pca(self):
    scale_cols = ['year','engineSize','litre','tax','mileage','mpg',' 8 Series',' California', ' R8',' X7',' i3',' i8']
    self.data_pca=self.data.copy()
    self.y = self.data_pca.pop('price')
    self.data_pca[scale_cols] = self.scaler.fit_transform(self.data_pca[scale_cols])
    self.pca=PCA(n_components=53)
    self.X_pca=self.pca.fit_transform(self.data_pca)
    np.cumsum(np.round(self.pca.explained_variance_ratio_,decimals=4)*100)[0:10]
    self.X_train_pca,self.X_test_pca,self.y_train,self.y_test=train_test_split(self.X_pca,self.y,test_size=0.2,random_state=2)
    model=LinearRegression()
    model.fit(self.X_train_pca,self.y_train)
    self.y_pred_pca = model.predict(self.X_test_pca)
    return r2_score(self.y_test,self.y_pred_pca)
```

Plot Methodu

Bu fonksiyonda y_test ve y_pred'e göre scatter grafiği oluşturuldu.

```
def plot(self):
    plt.subplots(figsize=(12,8))
    plt.scatter(range(len(self.y_test)), self.y_test, color='blue')
    plt.scatter(range(len(self.y_pred)), self.y_pred, color='red')
    plt.show()
```

Main fonksiyonunda LinearR sınıfından bir nesne oluşturuldu ve ekrana çıkan sonuçlar bastırıldı.

```
def main():
    prediction_price = LinearR(cars)
    prediction_price.feature_engineering(cars)
    prediction_price.data_visualization()
    prediction_price.dummy()
    prediction_price.feature_delete()
    prediction_price.train_test()
    prediction_price.min_max_scaler()
    prediction_price.linear_regression()
    prediction_price.X_train_new=prediction_price.ols_build(prediction_price.X_train_rfe,prediction_price.y_train)
    prediction_price.ols_error()
    prediction_price.regression_pred_test()
    print(f"R'2:{prediction_price.r2_score()}")
    print(f"Katsayılar:{prediction_price.coefficients()}")
    print(f"RMSE:{prediction_price.rmse()}")
    print(f"PCA R'2:{prediction_price.pca()}")

main()
```

Logistic Regression

Verilen datasetini python veri dosyasında okundu. Nesne yönelimli programlama prosedürüne uymak açısından LogisticR adında bir sınıf oluşturuldu. Bu sınıfa belirli methodlar eklendi.

Feature_Engineering Methodu

Bu fonksiyon da mpg veriseti *0.43 ile çarpıldı. Bulunan değer mileage ile çarpıldı. Ardından 100'e bölünerek litre adında bir feature oluşturuldu.

```
import numpy as np
import pandas as pd
import seaborn as sn
cars = pd.read_csv('cars_dataset.csv')
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.neighbors import LocalOutlierFactor
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import r2_score
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
import warnings
warnings.simplefilter(action='ignore', category=Warning)

class LogisticR():

    def __init__(self,data):
        self.data = cars[cars["Make"] == data]
    def feature_engineering(self):
        self.litre = self.data.insert(2, 'litre', pd.Series(self.data["mileage"]*(self.data["mpg"]*0.43))/100)
        self.litre = self.data["litre"]
        return self.litre
```

Dummy, Data Visualization ve Outlier Methodları

Dummy methodunda, transmission, fuelType kategorik verileri get_dummies methodu ile dönüştürüldü ve dataset güncellendi. Data_visualization methodunda, fuelType ve year'in price için grafikleri oluşturuldu. Outlier methodunda ise standart sapma ve ortalamaya göre aykırı değerler hesaplandı. Datasetten çıkartıldı.

```
def dummy(self):
    self.data =pd.get_dummies(self.data,columns=['transmission','fuelType'],drop_first=True)
def data_visualization(self):
    self.data['Make'].str.strip()
    plt.figure(figsize=(10,8))
    sn.boxplot(x=self.data["fuelType"], y=self.data["price"])
    plt.show()
    sn.regplot(x=self.data["year"], y=self.data["price"])
    plt.show()

def outlier(self):
    for column in self.data.columns[1:-8]:
        for spec in self.data["Make"].unique():
            selected_spec = self.data[self.data["Make"]==spec]
            selected_column = selected_spec[column]
            std = selected_column.std()
            avg = selected_column.mean()
            three_sigma_plus=avg + (3*std)
            three_sigma_minus=(  (3*std)-avg)
            outliers=selected_column[((selected_spec[column] > three_sigma_plus) | (selected_spec[column] < three_sigma_minus))].index
            self.data.drop(index=outliers, inplace=True)
    return self.data
```

Local Outlier ve Min Max Scaler Methodları

Local outlier methodunda, komşulara göre aykırı değerler tespit edilmektedir. Ardından eşik değer ve aykırı değerler bulundu. Dataset güncellenmiş oldu. Min_max_scaler methodunda ise, datasetteki sütun değerlerinin baskınlığını azaltmak için MinMaxScaler yöntemi uygulandı.

```
def local_outlier(self):
    self.y = self.data['model']
    self.data.drop(['model'],axis=1,inplace=True)
    self.data.drop(['Make'],axis=1,inplace=True)
    self.clf=LocalOutlierFactor(n_neighbors=20,contamination=0.1)
    self.clf.fit_predict(self.data)
    self.car_new_scores= self.clf.negative_outlier_factor_
    np.sort(self.car_new_scores)[0:3]
    self.esik_deger=np.sort(self.car_new_scores)[2]
    self.aykiri_tf = self.car_new_scores > self.esik_deger
    self.baski_degeri = self.data[self.car_new_scores == self.esik_deger]
    self.aykirilar = self.data[~self.aykiri_tf]
    self.res=self.aykirilar.to_records(index=False)
    self.res[:]=self.baski_degeri.to_records(index=False)
    self.data[~self.aykiri_tf] = pd.DataFrame(self.res,index=self.data[~self.aykiri_tf].index)
    return self.data[~self.aykiri_tf]

def min_max_scaler(self):
    self.scaler = MinMaxScaler()
    self.cols=self.data.columns
    self.data = self.scaler.fit_transform(self.data)
    self.data = pd.DataFrame(self.data, columns=[self.cols])
    return self.data
```

Train Test, Logistic Regression, Accuracy Score ve PCA Methodları

Train_test methodunda dataset, pca için eğitmek ve test etmek için ayrıldı. Logistic_regression methodunda, model oluşturuldu. Fit edilip tahmin yapıldı. Accuracy_score methodunda, y_test ve y_pred'e bağlı accuracy score ekrana getirildi. Pca methodunda ise, PCA'yı etkileyecek değer girildi ve model tekrardan oluşturulup fit edildi. Dataset train-test için ayrıldı.

```
def train_test(self):
    self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(self.data, self.y, test_size=0.2, random_state=2)
def logistic_regression(self):
    self.logreg = LogisticRegression(random_state=0,max_iter=120,multi_class='multinomial', solver='lbfgs')
    self.logreg.fit(self.X_train, self.y_train)
    self.y_pred = self.logreg.predict(self.X_test)

def accuracy_score(self):
    return accuracy_score(self.y_test, self.y_pred)
def pca(self):
    self.pca=PCA(n_components=7)
    self.data_pca=self.data.copy()
    self.X_pca=self.pca.fit_transform(self.data_pca)
    np.cumsum(np.round(self.pca.explained_variance_ratio_,decimals=4)*100)[0:10]
    self.X_train_pca,self.X_test_pca,self.y_train,self.y_test=train_test_split(self.X_pca,self.y,test_size=0.2,random_state=2)
    self.model=LogisticRegression(max_iter=1000)
    self.model.fit(self.X_train_pca,self.y_train)
```

Confusion Matrix ve Pca Score Methodları

Confusion_matrix methodunda, y_test ve y_pred değişkenlerine göre confusion matrix'i oluşturulup ekrana basıldı. Pca_score methodunda ise X_Test_pca ve y_test'e göre pca score'u hesaplandı.

```
def confusion_matrix(self):
    self.cm=confusion_matrix(self.y_test,self.y_pred)
    plt.figure(figsize=(20,10))
    sn.heatmap(self.cm,annot=True)
    plt.show()
def pca_score(self):
    return self.model.score(self.X_test_pca,self.y_test)
```

Car fonksiyonu tanımlandı. Burada sınıf oluşturuldu ve methodlar çağırıldı. Main fonksiyonunda ise tüm arabalar isimlerine göre tahminlerin oluşması sağlandı.

```
def Car(car_name="audi"):
    car=LogisticR(car_name)
    car.feature_engineering()
    car.data_visualization()
    car.dummy()
    car.outlier()
    print(car.local_outlier())
    car.min_max_scaler()
    car.train_test()
    car.logistic_regression()
    car.confusion_matrix()
    print("Modelin Accuracy Score'u: {0:0.4f}".format(car.accuracy_score()))
    car.pca()
    print("Modelin PCA Score'u: {0:0.4f}".format(car.pca_score()))


def main():
    Car("audi")
    Car("BMW")
    Car("Ford")
    Car("vw")
    Car("toyota")
    Car("skoda")
    Car("Hyundai")
main()
```