YAZ304
Sinir Ağları ve Makine Öğrenmesi Dönem Projesi
Kaan İnce 210911216

Home Price Prediction

Kaggle'dan indirilen data seti okundu. Nesne yönelimli programlama prosedürüne uymak açısından HomePricePrediction adında bir sınıf oluşturuldu. Bu sınıfa belirli methodlar eklendi.

FeatureDrop ve MissingValueCheck Methodları

FeatureDrop methodunda, datasette olan date,country,street ve statezip featureları çıkartıldı. MissingValueCheck methodunda, info() komutu çağırıldı ve verilerde boş değer var mı yok mu kontrol edildi.

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
import math
from scipy import stats
from scipy.stats import skew
from sklearn.neighbors import LocalOutlierFactor

from sklearn.preprocessing import RobustScaler,StandardScaler,MinMaxScaler
from sklearn.linear_model import LinearRegression,Ridge,Lasso,ElasticNet
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.base import clone
#XGBoost
import xgboost as xgb
#warning
import warnings
warnings.filterwarnings('ignore')

data = pd.read_csv('home.csv',na_values="?",comment="\t",skipinitialspace=True)
home=data.copy()
class HousePricePrediction():
    def __init__(self,data):
        self.data=data
    def featureDrop(self):
        f=["date","country","street","statezip"]
        for i in f:
            self.data.drop([i], axis = 1, inplace = True)
    def missingValueCheck(self):
        return self.data.info()
```

EDA Methodu

EDA methodunda, datasetin clustermap ve boxplot grafikleri çağrıldı. Verideki aykırı değerler tespit edildi.

```
def EDA(self):
    corr_matrix = self.data.corr()
    sns.clustermap(corr_matrix, annot = True,fmt=".2f")
    plt.title("Correlation btw features")
    plt.show()
    sns.pairplot(self.data,diag_kind="kde",markers="+")
    plt.show()
    for c in self.data.columns:
        plt.figure()
        sns.boxplot(x=c,data=self.data,orient="v")
        plt.show()
```

LocalOutlier ve Outlier Methodu

Local outlier methodunda, komşulara göre aykırı değerler tespit edildi. Ardından eşik değer ve aykırı değerler bulundu. Dataset güncellenmiş oldu. Outlier methodunda ise standart sapma ve ortalamaya göre aykırı değerler hesaplandı. Datasetten çıkartıldı.

```
def localOutlier(self):
    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)
def outlier(self):
    for column in self.data.columns[1:-8]:
        for spec in self.data["price"].unique():
            selected_spec = self.data[self.data["price"]==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
```

FeatureEngineeringSkew ve FeatureEngineeringEncoding Methodları

FeatureEngineeringSkew methodunda, price dağılımları ekrana bastırıldı. sqft_lot,sqft_living,sqft_above,price featurelarındaki çarpıklıklar düzeltildi. Düzeltilmiş price feature'ın dağılımı tekrardan ekrana bastırıldı.FeatureEngineeringEncoding methodunda ise floors ve city feature'ları dönüştürüldü ve yr_renovated feature'u datasetten çıkartıldı.

```
def featureEngineeringSkew(self):
    #feature çarpıklıkları
    plt.figure()
    stats.probplot(self.data["price"],plot=plt)
    plt.show()
    skewed_up=["sqft_lot","sqft_living","sqft_above","price"]
    for i in skewed_up:
        self.data[i]=np.log(self.data[i])
    plt.figure()
    stats.probplot(self.data["price"],plot=plt)
    plt.show()
    #feature
    skewed_features=self.data.apply(lambda x:skew(x.dropna())).sort_values(ascending=False)
    skewness=pd.DataFrame(skewed_features,columns=["skewed"])
    print(skewness)
def featureEngineeringEncoding(self):
    dummies_list=['floors','city']
    for i in dummies_list:
        temp_df = pd.get_dummies(eval('home' + '.' + i), drop_first=True)
        self.data = pd.concat([self.data, temp_df], axis=1)
        self.data.drop([i], axis=1, inplace=True)
    self.data.drop(["yr_renovated"], axis = 1, inplace = True)
```

TrainTestSplit,Standardization ve LinearRegression Methodları

TrainTestSplit methodunda dataset, eğitmek ve test etmek için ayrıldı. Standardization methodunda featurelar baskınlığı azaltmak için 0-1 değerleri arasına getirildi. LinearRegression methodunda ise model fit edilip tahmin yapıldı. MSE ve R^2 score ekrana bastırıldı.

```
def trainTestSplit(self):
    self.x=self.data.drop(["price"],axis=1)
    self.y=self.data["price"]
    self.X_train,self.X_test,self.Y_train,self.Y_test=train_test_split(self.x,self.y, train_size = 0.7, test_size = 0.3, random_state = 42)
def standardization(self):
    scaler = StandardScaler()
    self.X_train=scaler.fit_transform(self.X_train)
    self.X_test=scaler.transform(self.X_test)
def linearRegression(self):
    self.lr=LinearRegression()
    self.lr.fit(self.X_train,self.Y_train)
    print(f"LR COEF:{self.lr.coef_}")
    self.y_predicted_dummy=self.lr.predict(self.X_test)
    print(f"Linear R2_Score: {r2_score(self.Y_test,self.y_predicted_dummy)}")
    mse=mean_squared_error(self.Y_test,self.y_predicted_dummy)
    print(f"Linear Mean Squared Error {mse}")
```

RidgeRegression ve LassoRegression Methodları

RidgeRegression ve LassoRegression methodlarında, alpha değer aralıkları girildi. GridSearch ile en iyi alpha değerleri tespit edildi. Model Fit edilip tahmin yapıldı. Ardından Coef, MSE ve R^2 Score'u ekrana bastırıldı.

```
def ridgeRegression(self):
    ridge=Ridge(random_state=42,max_iter=10000)
    self.alphas = np.logspace(-4,-0.5,30)
    tuned_parameters=[{'alpha':self.alphas}]
    self.n_folds=5
    self.RidgeClf=GridSearchCV(ridge,tuned_parameters,cv=self.n_folds,scoring="neg_mean_squared_error",refit=True)
    self.RidgeClf.fit(self.X_train,self.Y_train)
    scores=self.RidgeClf.cv_results_["mean_test_score"]
    scores_std=self.RidgeClf.cv_results_["std_test_score"]
    print("Ridge Coef:", self.RidgeClf.best_estimator_.coef_)
    self.y_predicted_dummy_ridge = self.RidgeClf.predict(self.X_test)
    mse = mean_squared_error(self.Y_test,self.y_predicted_dummy_ridge)
    print(f"Ridge MSE {mse}")
    print(f" Ridge R2_Score: {r2_score(self.Y_test,self.y_predicted_dummy_ridge)}")

def lassoRegression(self):
    self.lasso=Lasso(random_state=42,max_iter=10000)
    tuned_parameters=[{'alpha':self.alphas}]
    self.LassoClf=GridSearchCV(self.lasso,tuned_parameters,cv=self.n_folds,scoring="neg_mean_squared_error",refit=True)
    self.LassoClf.fit(self.X_train,self.Y_train)
    scores=self.LassoClf.cv_results_["mean_test_score"]
    scores_std=self.LassoClf.cv_results_["std_test_score"]
    print("Lasso Coef:", self.LassoClf.best_estimator_.coef_)
    self.y_predicted_dummy_lasso = self.LassoClf.predict(self.X_test)
    mse = mean_squared_error(self.Y_test,self.y_predicted_dummy_lasso)
    print(f"Lasso MSE {mse}")
    print(f" Lasso R2_Score: {r2_score(self.Y_test,self.y_predicted_dummy_lasso)}")
```

ElasticNetRegression ve XGBoost Methodları

ElasticNetRegression methodunda, alpha değer aralıkları girildi. GridSearch ile en iyi alpha değerleri tespit edildi. Model Fit edilip tahmin yapıldı. Ardından Coef, MSE ve R^2 Score'u ekrana bastırıldı. XGBoost methodunda ise model oluşturuldu. Fit edilip tahmin yapıldı. MSE ve R^2 Score'u ekrana bastırıldı.

```
def elasticNetRegression(self):
    self.alphas = np.logspace(-4,-0.5,30)
    parametersGrid={"alpha":self.alphas,"l1_ratio":np.arange(0.0,1.0,0.05)}
    self.eNet=ElasticNet(random_state=42,max_iter=10000)
    self.eNetClf=GridSearchCV(self.eNet,parametersGrid,cv=self.n_folds,scoring="neg_mean_squared_error",refit=True)
    self.eNetClf.fit(self.X_train,self.Y_train)
    print("ElasticNet Coef:", self.eNetClf.best_estimator_.coef_)
    self.y_predicted_dummy_eNet = self.eNetClf.predict(self.X_test)
    mse = mean_squared_error(self.Y_test,self.y_predicted_dummy_eNet)
    print(f"ElasticNet MSE {mse}")
    print(f" ElasticNet R2_Score: {r2_score(self.Y_test,self.y_predicted_dummy_eNet)}")
def XGBoost(self):
    self.model_xgb=xgb.XGBRegressor(n_estimators=100, learning_rate=0.08, gamma=0, subsample=0.75,
                    colsample_bytree=1, max_depth=7)
    self.model_xgb.fit(self.X_train,self.Y_train)
    self.y_predicted_dummy_xgb=self.model_xgb.predict(self.X_test)
    mse = mean_squared_error(self.Y_test,self.y_predicted_dummy_xgb)
    print(f"XGB MSE {mse}")
    print(f" XGB R2_Score: {r2_score(self.Y_test,self.y_predicted_dummy_xgb)}")
```

AveragingModels ve ViewData Methodu

AveragingModels methodunda, en iyi skora sahip modellerin ortalaması alındı ve en yüksek tahmin sonucu olan 0.76 (Kaggle'da gördüğüm kadarıyla en yükseği) elde edildi. ViewData methodunda, datanın son hali ekrana getirildi.

```
def AveragingModels(self):
    self.models=(self.RidgeClf,self.lr)
    self.models_=[clone(x) for x in self.models]
    for model in self.models_:
        model.fit(self.X_train,self.Y_train)
    predictions=np.column_stack([model.predict(self.X_test) for model in self.models_])
    self.y_predicted_dummy_averaging= np.mean(predictions,axis=1)
    mse = mean_squared_error(self.Y_test,self.y_predicted_dummy_averaging)
    print(f"Averaging Models MSE {mse}")
    print(f" Averaging Models R2_Score: {r2_score(self.Y_test,self.y_predicted_dummy_averaging)}")

def viewData(self):
    return self.data
```

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

```
def main():
    h = HousePricePrediction(home)
    h.featureDrop()
    print(h.missingValueCheck())
    h.EDA()
    h.featureEngineeringEncoding()
    h.localOutlier()
    h.outlier()
    h.featureEngineeringSkew()
    h.trainTestSplit()
    h.standardization()
    h.linearRegression()
    h.ridgeRegression()
    h.lassoRegression()
    h.elasticNetRegression()
    h.XGBoost()
    h.AveragingModels()
    #print(h.viewData())
main()
```