CSE 385: Data Mining

Project

Submitted to:

Dr. Mahmoud Mounir

Eng. Ahmed Hesham

Submitted by:

Engy Samy Salah 16P3004

Mayar Wessam Hassan 16P3008

Sandro Nael 16P3000

Yara Hossam Mohamed 16P3002

Our projects consist of 4 types of classes which are: preprocessing, classifier, regression and clustering.

1. Class Preprocessing:

firstly, we pass the data set to this class to perform a lot of functions such as:

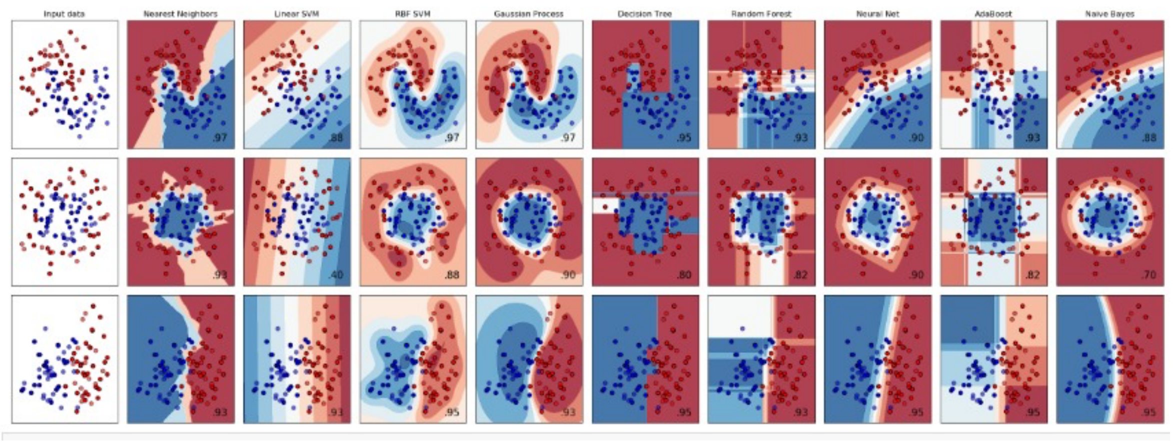
- def missing_data(self,X_train): which removes all missing rows and columns from the data set
- def encode (self,attribute,X_train): which encodes the nominal data
- def scale (self,method,X_train): which resize the large or small data; we have "StandardScaler()" and "MinMaxScaler()" and "MaxAbsScaler()"

Then, we apply the classifier and regression and clustering on the x_train y_train x_test and y_test

2. *Class Classifier:*

we pass the data set to it (which is ”wisconsin breast cancer”)

the init() function has multiple types of classifiers which are:

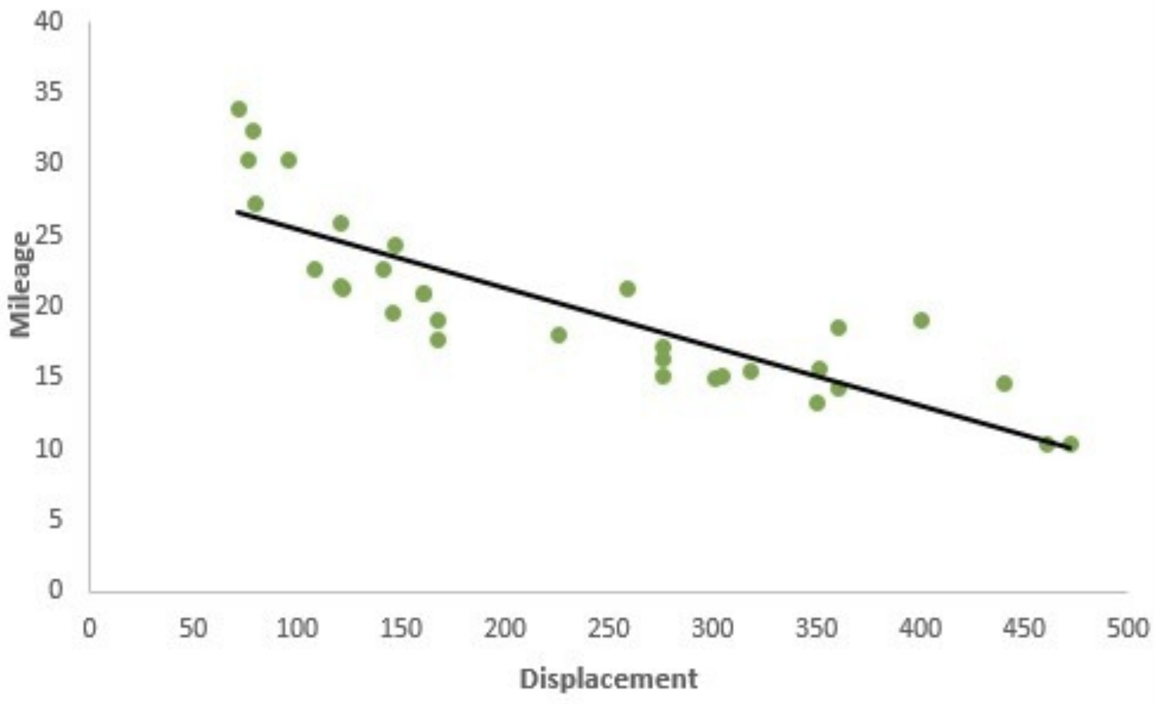


1. **KNeighborsClassifier :** operates by checking the distance from some test example to the known values of some training example. The group of data points/class that would give the smallest distance between the training points and the testing point is the class that is selected.
2. **RandomForestClassifier:** Once the network has divided the data down to one example, the example will be put into a class that corresponds to a key. When multiple random forest classifiers are linked together
3. **DecisionTreeClassifier:** by breaking down a dataset into smaller and smaller subsets based on different criteria. Different sorting criteria will be used to divide the dataset, with the number of examples getting smaller with every division.
4. **BayesianClassifier:** determines the probability that an example belongs to some class, calculating the probability that an event will occur given that some input event has occurred. When it does this calculation, it is assumed that all the predictors of a class have the same effect on the outcome, that the predictors are independent

3. Class Regression:

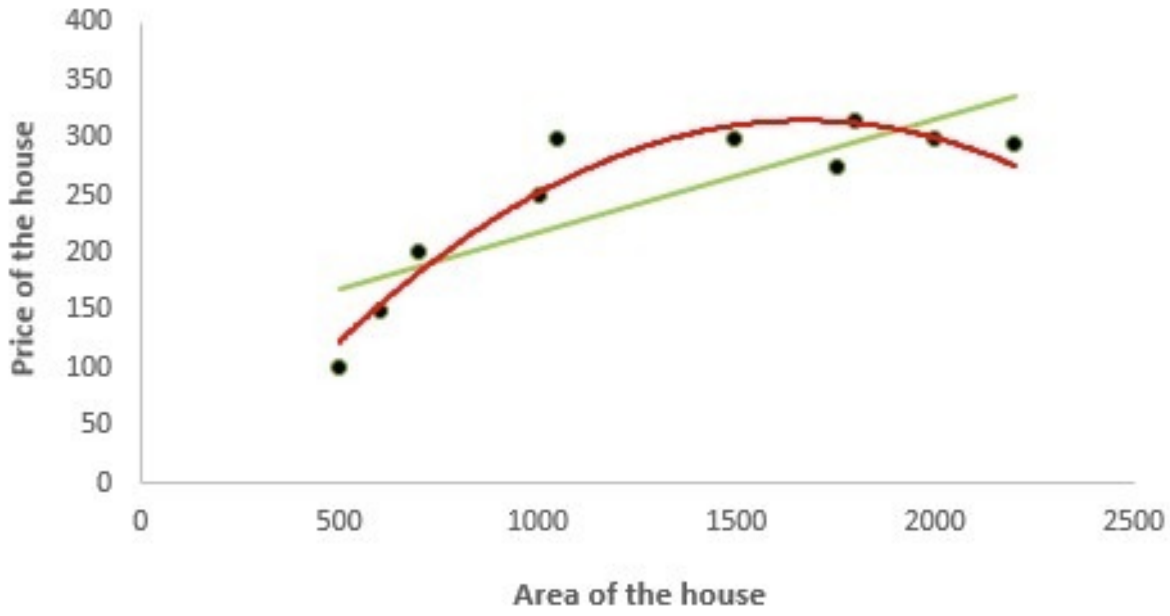
the init() function has multiple types of regressors which are:

1. LinerRegression:



It is the simplest form of regression. It is a technique in which the dependent variable is continuous in nature. The relationship between the dependent variable and independent variables is assumed to be linear in nature. We can observe that the given plot represents a somehow linear relationship between the mileage and displacement of cars. The green points are the actual observations while the black line fitted is the line of regression

2. **PolynomialRegression :**



It is a technique to fit a nonlinear equation by taking polynomial functions of independent variable.

In the figure given below, you can see the red curve fits the data better than the green curve. Hence in the situations where the relation between the dependent and independent variable seems to be non-linear we can deploy Polynomial Regression Models.

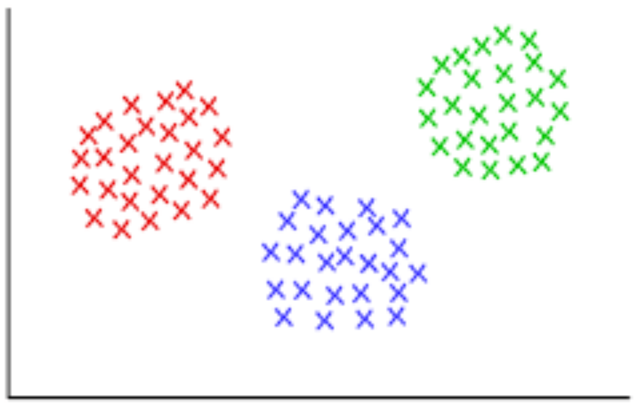
3. DecisionTreeRegression :

builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy), each representing values for the attribute tested. Leaf node (e.g., Hours Played) represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data

4. __KnnRegressor :__

Regression based on k-nearest neighbors. The target is predicted by local interpolation of the targets associated of the nearest neighbors in the training set.

4. ***Class Clustering:***



is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. In simple words, the aim is to segregate groups with similar traits and assign them into clusters.

Code:

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import MaxAbsScaler
from sklearn.preprocessing import Imputer
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.naive_bayes import GaussianNB
from sklearn.cluster import KMeans
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
class preprocessing :
    def scale (self,method,X_train):
        if(method=="standardscaler"):
            stdsc = StandardScaler()
            X_transformed = stdsc.fit_transform(X_train)
        if(method=="minmaxscaler"):
            mms = MinMaxScaler()
            X_transformed = mms.fit_transform(X_train)
        if(method=="maxscaler"):
            max_abs_scaler = preprocessing.MaxAbsScaler()
            X_transformed = max_abs_scaler.fit_transform(X_train)
        if(method=="minscaler"):
            max_abs_scaler = preprocessing.MaxAbsScaler()
            X_transformed = max_abs_scaler.fit_transform(X_train)
        return X_transformed
    def encode (self,attribute,X_train):
        le = LabelEncoder()
        y= le.fit_transform(X_train[attribute].values)
        return y
    def encodey (self,y):
        le = LabelEncoder()
        y= le.fit_transform(y.values)
        return y
    def missing_data(self,X_train):
        X_train.dropna()
        return X_train
class classifier :
    def _init_(self, method,X_train,X_test,y_train,y_test):
        if(method=="knn"):
            knn = KNeighborsClassifier(n_neighbors=2)
            knn.fit(X_train, y_train)
            acc=knn.score(X_test,y_test)
            pred = knn.predict(X_test)
            print(acc)
            print(pred)

        if(method=="RandomForestClassifier"):
            clf=RandomForestClassifier(n_estimators=80)##when number of
estimators dec acc inc###
            clf.fit(X_train,y_train)
            acc= clf.score(X_test,y_test)
            pred=clf.predict(X_test)
            print(acc)
            print(pred)
        if(method=="DecisionTreeClassifier"):
            clf= tree.DecisionTreeClassifier()
            clf.fit(X_train,y_train)
            acc = clf.score(X_test,y_test)
            pred=clf.predict(X_test)
            print(acc)
            print(pred)
        if(method=="BayesianClassifier"):
            clf = GaussianNB()
            clf.fit(X_train,y_train)
            acc = clf.score(X_test,y_test)
            pred=clf.predict(X_test)
            print(acc)
            print(pred)

class regression :
    def _init_(self, method,X_train,X_test,y_train,y_test):
        if(method=="LinerRegression"):
            lr = LinearRegression(normalize=True)
            lr.fit(X_train,y_train)
            acc = lr.score(X_test,y_test)
            predictions=lr.predict(X_test)
            print(acc)
            print(predictions)
        if(method=="PolynomialRegression"):
            poly = PolynomialFeatures(2)
            X_poly = poly.fit_transform(X_train)
            poly_model = LinearRegression()
            poly_model.fit(X_poly, y_train)
            pred = poly_model.predict(X_poly)
            print(acc)
        if(method=="DecisionTreeRegression"):
            regressor = DecisionTreeRegressor(random_state = 0)
            regressor.fit(X_train,y_train)
            pred = regressor.predict(X_test)
            acc = regressor.score(X_test,y_test)
            print(acc)
            print(pred)
        if(method=="KnnRegressor"):
            neigh = KNeighborsRegressor(n_neighbors=5) ##when k increase the
acc inc##
            neigh.fit(X_train, y_train)
            acc=neigh.score(X_test,y_test)
            pred = neigh.predict(X_test)
            print(acc)
            print(pred)
class clustering :
    def _init_(self,X_train,X_test,y_train,y_test):
            kmeans = KMeans(n_clusters=2, random_state=0)
            kmeans.fit(X_train,y_train)
            pred=kmeans.predict(X_test)
            print(pred)
X = pd.read_csv('F:\College\Semester 7\Computer Vision\diamonds.csv')
y=X.pop('price')
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.33)
```

```
####preprocessing######
preprocess = preprocessing()
def preproces(x):
    z=preprocess.encode('cut',x)
    x=x.assign(cut=z)
    z=preprocess.encode('color',x)
    x=x.assign(color=z)
    z=preprocess.encode('clarity',x)
    x=x.assign(clarity=z)
    x.isnull().sum()
    x=preprocess. missing_data(x)
    z=preprocess.scale("minmaxscaler",x)
    x=z
    return x
def preprocesy(y):
    y.isnull().sum()
    y=preprocess. missing_data(y)
    return y
X_train=preproces(X_train)
y_train=preprocesy(y_train)
X_test=preproces(X_test)
y_test=preprocesy(y_test)
regres=regression("KnnRegressor",X_train,X_test,y_train,y_test)
#######classification####
X = pd.read_csv('F:\College\Semester 7\Computer Vision\data.csv')
y=X.pop('diagnosis')
X.pop('Unnamed: 32')
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.33)
```

```
def preproces1(x):
    x.isnull().sum()
    x=preprocess.missing_data(x)
    z=preprocess.scale("minmaxscaler",x)
    x=z
    return x
def preproces2(y):
    y.isnull().sum()
    y=preprocess. missing_data(y)
    y=preprocess.encodey (y)
    return y
X_train=preproces1(X_train)
y_train=preproces2(y_train)
X_test=preproces1(X_test)
y_test=preproces2(y_test)
classification=classifier("BayesianClassifier",X_train,X_test,y_train,y_test)
#######cluster####
X = pd.read_csv('F:\College\Semester 7\Computer Vision\Iris.csv')
y=X.pop('Species')
#X.pop('Unnamed: 32')
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.33)
def preproces1(x):
    x.isnull().sum()
    x=preprocess.missing_data(x)
    z=preprocess.scale("minmaxscaler",x)
    x=z
    return x
def preproces2(y):
    y.isnull().sum()
    y=preprocess. missing_data(y)
    y=preprocess.encodey (y)
    return y
X_train=preproces1(X_train)
y_train=preproces2(y_train)
X_test=preproces1(X_test)
y_test=preproces2(y_test)
cluster=clustering(X_train,X_test,y_train,y_test)
```