BIS634 - Assignment5

Name: Zhiyuan Cao; NetID: zc347

Exercise 1

Implementation of quad-tree

First I write the code for quad-tree to store the 2-dimensional data. Basically the class `QuadTree` contains the following functions:

- `__init__`
- `get_descendant_count`
- `get_data_in_range`
- `__repr__`

Please see my code for details.

KNN

Given a new (x, y) point and a value of k (the number of nearest neighbors to examine), my implementation is able to identify the most common class within those k nearest neighbors. Basically the class `KNN` contains the following functions:

- `in_box`
- `dist`
- `find_R`
- `within_d`
- `points_in_d`
- `predict`

Please see my code for details.

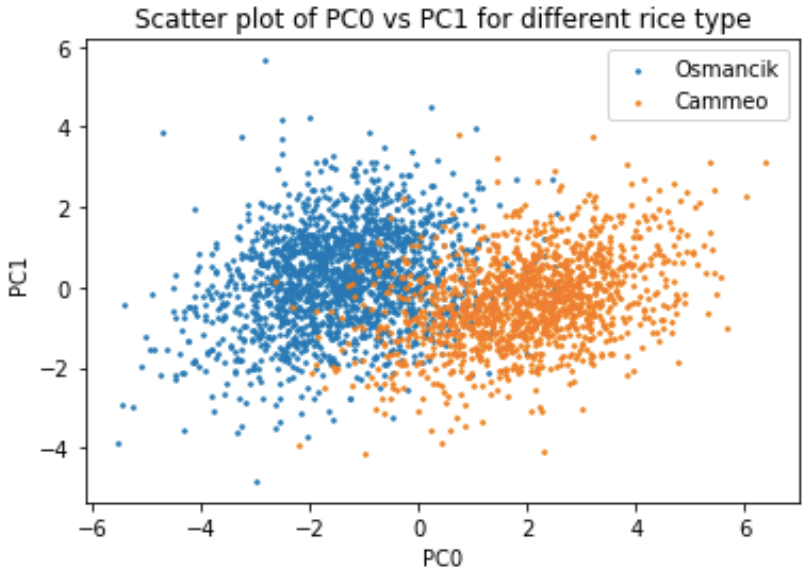
Data normalization

I obtain and normailze my data by the following code:

```
def acquire_data(t_size=.8):
    data = pd.read_excel('Rice_Cammeo_Osmancik.xlsx')
    X_train, X_test, y_train, y_test = train_test_split(data.drop(['Class'], axis=1),
data['Class'], train_size=t_size)
    X_train = pd.DataFrame(StandardScaler().fit_transform(X_train), columns =
X_train.columns)
    X_test = pd.DataFrame(StandardScaler().fit_transform(X_test), columns =
X_test.columns)
    pca = decomposition.PCA(n_components=2)
    X_train_pca = pca.fit_transform(X_train)
    X_train_pc0 = X_train_pca[:, 0]
```

```
    X_train_pc1 = X_train_pca[:, 1]
    train = list(zip(X_train_pc0, X_train_pc1, y_train))
    X_test_pca = pca.transform(X_test)
    X_test_pc0 = X_test_pca[:, 0]
    X_test_pc1 = X_test_pca[:, 1]
    test = list(zip(X_test_pc0, X_test_pc1, y_test))
    return train, test
```

Scatterplot



This is the scatter plot for PC0 vs. PC1 for different rice type.

Comment: From this figure, we can observe that the two clusters 'Osmancik' and 'Cammeo' are away from each other, since we can clearly tell the blue points from the orange ones. Therefore it is effective to use KNN to predict the type of rice.

Train-test split

I do train-test split as follows:

```
def acquire_data(t_size=.8):
    data = pd.read_excel('Rice_Cammeo_Osmancik.xlsx')
    X_train, X_test, y_train, y_test = train_test_split(data.drop(['Class'], axis=1),
data['Class'], train_size=t_size)
    X_train = pd.DataFrame(StandardScaler().fit_transform(X_train), columns =
X_train.columns)
    X_test = pd.DataFrame(StandardScaler().fit_transform(X_test), columns =
X_test.columns)
    pca = decomposition.PCA(n_components=2)
    X_train_pca = pca.fit_transform(X_train)
    X_train_pc0 = X_train_pca[:, 0]
    X_train_pc1 = X_train_pca[:, 1]
    train = list(zip(X_train_pc0, X_train_pc1, y_train))
    X_test_pca = pca.transform(X_test)
```

```
    X_test_pc0 = X_test_pca[:, 0]
    X_test_pc1 = X_test_pca[:, 1]
    test = list(zip(X_test_pc0, X_test_pc1, y_test))
    return train, test
```

I split the dataset by 80% vs. 20%. Then I normalize the data.

Confusition matrix

```
In [108]:
1 nevermind1, nevermind2, y_true = zip(*test_data)
2 y_pred = [KNN_model.predict(test_data[i][0], test_data[i][1], 1) for i in range(len(test_data))]
3 confusion_matrix(y_true, y_pred)

Out[108]: array([[292,  30],
                 [ 32, 408]])
```

This is the confusion matrix for k = 1.

```
In [109]:
1 nevermind1, nevermind2, y_true = zip(*test_data)
2 y_pred = [KNN_model.predict(test_data[i][0], test_data[i][1], 5) for i in range(len(test_data))]
3 confusion_matrix(y_true, y_pred)

Out[109]: array([[297,  25],
                 [ 24, 416]])
```

This is the confusion matrix for k = 5.

Interpretation：The confusion matrix shows the True Positive (TP), False Positive (FP), False Negative (FN) and True Negative (TN) of a classification model. From the results, we can see that he TP and TN of the model is greatly larger than FP and FN, which means the accuracy of the model is satisfactory. Besides, for k=5, the accuracy (number of TP and TN) is higher than the case when k=1, which is in line with our expectation.

Exercise 2

My data

This is the dataset I plan to use: https://www.kaggle.com/datasets/harikrishnareddyb/used-car-price-predictions. It is "Used car price predictions".

My analysis

Price	Year	Mileage	City	State	Vin	Make	Model
8995	2014	35725	El Paso	TX	19VDE2E53EE000083	Acura	ILX6-Speed
10888	2013	19606	Long Island City	NY	19VDE1F52DE012636	Acura	ILX5-Speed
8995	2013	48851	El Paso	TX	19VDE2E52DE000025	Acura	ILX6-Speed

I plan to analyze the relationship between price and other variables. In detail, I want to learn how the year, mileage, city, state, make and model of a vehicle influence its price. It is interesting because it is a classical machine learning problem. It can also be analyzed by SHAP plots to determine the influence of distinct variables.

Interactive & providing parameter

The year, mileage, city, state, make and model of a vehicle is interactive. Users can input the parameters of their own vehicles to predict the price of their vehicle.

Graphs

I can make a heatmap showing how year and mileage influence the price of a vehicle. I can also make bar plots on year vs. price, mileage vs. price, state vs. price, etc.

About website

My website mainly contains two parts: First is the visualization of the dataset. It contains the sheet of the data, its mean, median, std, min and max values. It also contains a US map which shows the price situation of different states. The second part is the prediction. User can input some parameters to predict the price of the given vehicle.

Challenge

The biggest challenge that I may have is:

- The machine learning algorithm: I will overcome it by trying different machine learning models, such as linear regression, random forest, xgboost, and select the model with the best performance as my prediction model.

Another challenge may be the design of website. I will overcome it by spending more time focusing on how to make the website more beautiful.

Exercise 3

Data source

The dataset for this exercise comes from https://statecancerprofiles.cancer.gov/incidencerates/index.php

Data cleaning

I perform data cleaning by the following code:

```
import pandas as pd

df = pd.read_csv('incd.csv')
df['State'] = df.apply(lambda x: x['State'][:-3], axis=1)
```

Besides, I convert all the columns with "data missing" or "N/A" into "nan" and I delete all the irrelevant rows manually. See my csv file "incd.csv" for details.

Server development

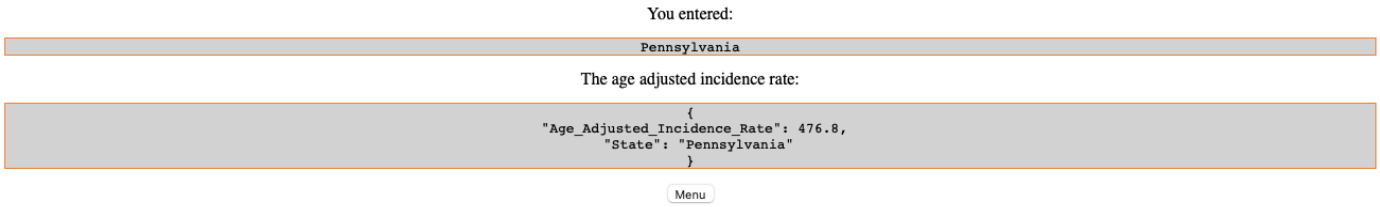
I write these three routes:

- `@app.route("/")`
- `@app.route("/state/<string:name>")`
- `@app.route("/info", methods=["GET"])`

Please see my code for details.

Below are some pictures for my website.

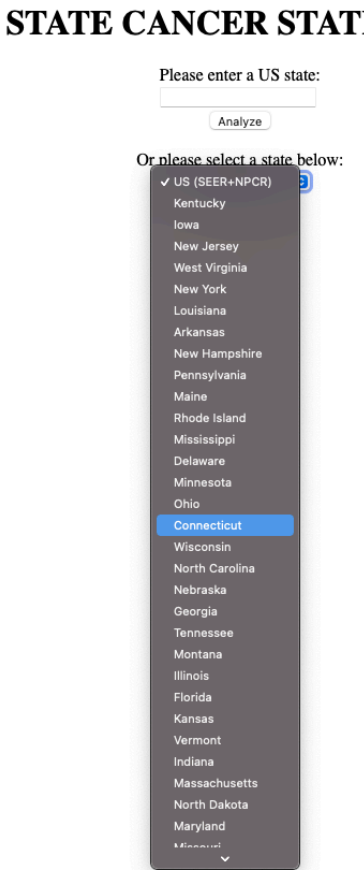
You can either enter a state name in the first box (Note that the first letter of a word should be capitalized.) or select a state from the drop down box. If the state enetered is correct, the result will be shown:



Otherwise, you will be directed to an error page.

Level-up function

As an extension function, I desing a drop-down menu so that the user can choose a state rather than typing it.



Appendix

Exercise 1

```
import numpy as np
import pandas as pd
import random
import statistics
import matplotlib.pyplot as plt
from sklearn import decomposition
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

class QuadTree:
    def __init__(self, data, bounding_box=None, max_leaf_data=3):
        if bounding_box is None:
            xs, ys, conditions = zip(*data)
            self.xlo = min(xs)
            self.ylo = min(ys)
            self.xhi = max(xs)*1.01
            self.yhi = max(ys)*1.01
        else:
            self.xlo = bounding_box['xlo']
```

```
            self.xhi = bounding_box['xhi']
            self.ylo = bounding_box['ylo']
            self.yhi = bounding_box['yhi']
        if len(data) <= max_leaf_data:
            self._data = data
            self.children = []
        else:
            self._data = None
            self.children = []
            xsplit = (self.xlo + self.xhi) / 2
            ysplit = (self.ylo + self.yhi) / 2
            bbox = [
                {'xlo': self.xlo, 'xhi': xsplit, 'ylo': self.ylo, 'yhi': ysplit},
                {'xlo': self.xlo, 'xhi': xsplit, 'ylo': ysplit, 'yhi': self.yhi},
                {'xlo': xsplit, 'xhi': self.xhi, 'ylo': self.ylo, 'yhi': ysplit},
                {'xlo': xsplit, 'xhi': self.xhi, 'ylo': ysplit, 'yhi': self.yhi}
            ]
            self.children = [
                QuadTree(self.get_data_in_range(data, my_bbox), my_bbox, max_leaf_data)
                for my_bbox in bbox
            ]

    def get_descendant_count(self):
        if not self.children:
            return len(self._data)
        else:
            return sum(child.get_descendant_count() for child in self.children)

    def get_data_in_range(self, data, bbox):
        result = []
        for x, y, condition in data:
            if bbox['xlo'] <= x < bbox['xhi'] and bbox['ylo'] <= y < bbox['yhi']:
                result.append([x, y, condition])
        return result

    def __repr__(self):
        return f'<QuadTree xlo={self.xlo} ylo={self.ylo} xhi={self.xhi} yhi={self.yhi}
#desc={self.get_descendant_count()}>'



class KNN():
    def __init__(self, data, max_leaf_data=3, k=5):
        self._QTree = QuadTree(data, max_leaf_data=max_leaf_data)

    def in_box(self, x, y, xlo, xhi, ylo, yhi):
        return (xlo <= x <= xhi) and (ylo <= y <= yhi)

    def dist(self, x1, y1, x2, y2):
```

```
        return np.sqrt((x1-x2)**2 + (y1-y2)**2)

    def find_R(self, x, y):
        if self.in_box(x, y, self._QTree.xlo, self._QTree.xhi, self._QTree.ylo,
self._QTree.yhi):
            Node = self._QTree
            while (Node._data is None):
                for i in range(4):
                    if self.in_box(x, y, Node.children[i].xlo, Node.children[i].xhi,
Node.children[i].ylo, Node.children[i].yhi):
                        Node = Node.children[i]
                        break
            return self.dist(Node.xlo, Node.ylo, Node.xhi, Node.yhi)
        else:
            return self.dist(self._QTree.xlo, self._QTree.ylo, self._QTree.xhi,
self._QTree.yhi) / 10

    def within_d(self, Node, x, y, d):
        if self.in_box(x, y, Node.xlo, Node.xhi, Node.ylo, Node.yhi):
            return True
        elif Node.xlo <= x <= Node.xhi:
            return min(abs(y-Node.ylo), abs(y-Node.yhi)) <= d
        elif self._QTree.ylo <= y <= self._QTree.yhi:
            return min(abs(x-Node.xlo), abs(x-Node.xhi)) <= d
        else:
            return min([
                self.dist(x, y, Node.xlo, Node.ylo),
                self.dist(x, y, Node.xlo, Node.yhi),
                self.dist(x, y, Node.xhi, Node.ylo),
                self.dist(x, y, Node.xhi, Node.yhi)
            ]) <= d

    def points_in_d(self, Node, x, y, d):
        if not Node.children:
            if self.within_d(Node, x, y, d):
                output = []
                for p in Node._data:
                    if self.dist(x, y, p[0], p[1]) <= d:
                        output.append(p)
                return output
            else:
                return []
        else:
            return self.points_in_d(Node.children[0], x, y, d) +
self.points_in_d(Node.children[1], x, y, d) + self.points_in_d(Node.children[2], x, y,
d) + self.points_in_d(Node.children[3], x, y, d)

    def predict(self, x, y, k):
        try:
```

```
            assert(k <= self._QTree.get_descendant_count())
        except:
            print("Error! k is too large and exceeds the number of dataset.")
        R = self.find_R(x, y)
        while len(self.points_in_d(self._QTree, x, y, R)) < k:
            R *= 2
        pts_within_R = self.points_in_d(self._QTree, x, y, R)
        dist_pts_within_R = [self.dist(x, y, a[0], a[1]) for a in pts_within_R]
        min_i = np.argpartition(dist_pts_within_R, k-1)[:k]
        pts = [pts_within_R[i] for i in min_i]
        return pd.DataFrame(pts)[2].value_counts().keys()[0]
```

```
def acquire_data(t_size=.8):
    data = pd.read_excel('Rice_Cammeo_Osmancik.xlsx')
    X_train, X_test, y_train, y_test = train_test_split(data.drop(['Class'], axis=1),
data['Class'], train_size=t_size)
    X_train = pd.DataFrame(StandardScaler().fit_transform(X_train), columns =
X_train.columns)
    X_test = pd.DataFrame(StandardScaler().fit_transform(X_test), columns =
X_test.columns)
    pca = decomposition.PCA(n_components=2)
    X_train_pca = pca.fit_transform(X_train)
    X_train_pc0 = X_train_pca[:, 0]
    X_train_pc1 = X_train_pca[:, 1]
    train = list(zip(X_train_pc0, X_train_pc1, y_train))
    X_test_pca = pca.transform(X_test)
    X_test_pc0 = X_test_pca[:, 0]
    X_test_pc1 = X_test_pca[:, 1]
    test = list(zip(X_test_pc0, X_test_pc1, y_test))
    return train, test
```

```
train_data, test_data = acquire_data()
KNN_model = KNN(train)
train_df = pd.DataFrame(train_data)
for rice in ['Osmancik', 'Cammeo']:
    df = train_df[train_df[2] == rice]
    plt.scatter(df[0], df[1], label=rice, s=2.5)
plt.xlabel('PC0')
plt.ylabel('PC1')
plt.title('Scatter plot of PC0 vs PC1 for different rice type')
plt.legend()
plt.show()
```

```
nevermind1, nevermind2, y_true = zip(*test_data)
y_pred = [KNN_model.predict(test_data[i][0], test_data[i][1], 1) for i in
range(len(test_data))]
confusion_matrix(y_true, y_pred)
```

```
nevermind1, nevermind2, y_true = zip(*test_data)
y_pred = [KNN_model.predict(test_data[i][0], test_data[i][1], 5) for i in
range(len(test_data))]
confusion_matrix(y_true, y_pred)
```

Exercise 3

```
from flask import Flask, render_template, request, jsonify
from collections import Counter
import pandas as pd

app = Flask(__name__)


@app.route("/")
def index():
    return render_template("index.html")

@app.route("/state/<string:name>")
def state(name):
    df = pd.read_csv('incd.csv')
    return jsonify(State=name, Age_Adjusted_Incidence_Rate = float(df[df['State'] ==
name]['Age-Adjusted Incidence Rate']))

@app.route("/info", methods=["GET"])
def info():
    name = request.args.get("state", None)
    df = pd.read_csv('incd.csv')
    if name in df['State'].value_counts().keys():
        return render_template("analyze.html",
analysis=state(name).get_data(as_text=True), name=name)
    else:
        return render_template("error.html")

if __name__ == "__main__":
    app.run(debug=True)
```

```
<html>
    <center>
    <body>
        <h1> ERROR: INVALID STATE NAME </h1>
        The state name is incorrect. Make sure the the first letter is capitalized.
        <form action="/" method="GET">
            <input type="submit" value="Menu">
        </form>
    </body>
    </center>
</html>
```

```
<html>
    <center>
    <body>
        You entered:
        <pre style="background-color: lightgray; margin-left: 5em; margin-right: 5em;
border: 1px solid rgb(255, 132, 0)">{{ name }}</pre>
        The age adjusted incidence rate:
        <pre style="background-color: lightgray; margin-left: 5em; margin-right: 5em;
border: 1px solid rgb(255, 132, 0)">{{ analysis }}</pre>
        <form action="/" method="GET">
            <input type="submit" value="Menu">
        </form>
    </body>
    </center>
</html>
```

```
<html>
    <center>
    <body>
        <h1> STATE CANCER STATISTICS </h1>
        Please enter a US state:
        <form action="/info" method="GET">
            <input type="text" name="state"></input>
            <br>
            <input type="submit" value="Analyze">
        </form>
        Or please select a state below:
        <form action="/info" method="GET">
            <select name="state" id="lang">
                <option value="US (SEER+NPCR)">US (SEER+NPCR)</option>
                <option value="Kentucky">Kentucky</option>
                <option value="Iowa">Iowa</option>
                <option value="New Jersey">New Jersey</option>
                <option value="West Virginia">West Virginia</option>
                <option value="New York">New York</option>
```

```
            <option value="Louisiana">Louisiana</option>
            <option value="Arkansas">Arkansas</option>
            <option value="New Hampshire">New Hampshire</option>
            <option value="Pennsylvania">Pennsylvania</option>
            <option value="Maine">Maine</option>
            <option value="Rhode Island">Rhode Island</option>
            <option value="Mississippi">Mississippi</option>
            <option value="Delaware">Delaware</option>
            <option value="Minnesota">Minnesota</option>
            <option value="Ohio">Ohio</option>
            <option value="Connecticut">Connecticut</option>
            <option value="Wisconsin">Wisconsin</option>
            <option value="North Carolina">North Carolina</option>
            <option value="Nebraska">Nebraska</option>
            <option value="Georgia">Georgia</option>
            <option value="Tennessee">Tennessee</option>
            <option value="Montana">Montana</option>
            <option value="Illinois">Illinois</option>
            <option value="Florida">Florida</option>
            <option value="Kansas">Kansas</option>
            <option value="Vermont">Vermont</option>
            <option value="Indiana">Indiana</option>
            <option value="Massachusetts">Massachusetts</option>
            <option value="North Dakota">North Dakota</option>
            <option value="Maryland">Maryland</option>
            <option value="Missouri">Missouri</option>
            <option value="South Dakota">South Dakota</option>
            <option value="Alabama">Alabama</option>
            <option value="Oklahoma">Oklahoma</option>
            <option value="Idaho">Idaho</option>
            <option value="Michigan">Michigan</option>
            <option value="South Carolina">South Carolina</option>
            <option value="Washington">Washington</option>
            <option value="Oregon">Oregon</option>
            <option value="Alaska">Alaska</option>
            <option value="District of Columbia">District of Columbia</option>
            <option value="Hawaii">Hawaii</option>
            <option value="Texas">Texas</option>
            <option value="Virginia">Virginia</option>
            <option value="Utah">Utah</option>
            <option value="Wyoming">Wyoming</option>
            <option value="California">California</option>
            <option value="Colorado">Colorado</option>
            <option value="Arizona">Arizona</option>
            <option value="New Mexico">New Mexico</option>
            <option value="Puerto Rico">Puerto Rico</option>
            <option value="Nevada">Nevada</option>
        </select> <br>
        <input type="submit" value="Analyze">
```

```
        </form>
    </body>
    </center>
</html>
```